Heinrich Heine
Universität
Düsseldorf

INSTITUT FÜR INFORMATIK
Softwaretechnik und Programmiersprachen
Universitätsstr. 1 D–40225 Düsseldorf

Entkopplung der Z3 Komponente in ProB mit ZeroMQ

Bachelorarbeit

vorgelegt von

Silas Alexander Kraume

22. Januar 2025

im Studiengang Informatik
zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

Erstgutachter: Prof. Dr. Michael Leuschel
Zweitgutachter: Dr. C. Bolz-Tereick

Selbstständigkeitserklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, den 22. Januar 2025

Silas Alexander Kraume

Zusammenfassung

Fassen Sie hier die Fragestellung, Motivation und Ergebnisse Ihrer Arbeit in wenigen Worten zusammen.

Die Zusammenfassung sollte den Umfang einer Seite nicht überschreiten.

Danksagung

Im Falle, dass Sie Ihrer Arbeit eine Danksagung für Ihre Unterstützer (Familie, Freunde, Betreuer) hinzufügen möchten, können Sie diese hier platzieren.

Dieser Part ist optional und kann im Quelltext auskommentiert werden.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Algorithmenverzeichnis

Quellcodeverzeichnis

1 Einführung

Die digitale Transformation hat unsere Welt grundlegend verändert und macht Software-Systeme zu einem unverzichtbaren Bestandteil des täglichen Lebens. Von sicherheitskritischen Anwendungen wie der Steuerung autonomer Fahrzeuge bis hin zu Finanzsystemen und medizinischen Geräten sind wir zunehmend auf Software angewiesen, die zuverlässig und fehlerfrei funktioniert. Die Gewährleistung von Korrektheit und Stabilität ist jedoch eine anspruchsvolle Aufgabe insbesondere angesichts der Komplexität moderner Systeme. Ein zentraler Baustein zur Bewältigung dieser Herausforderung ist der Einsatz von Modellierungs- und Verifikationswerkzeugen. Diese ermöglichen es, komplexe Systeme systematisch zu analysieren und sicherzustellen dass sie den gewünschten Spezifikationen entsprechen. Besonders hervorzuheben ist der Einsatz von SMT[1]-Solvern, die sich als leistungsfähige Werkzeuge etabliert haben, um schwierige logische Probleme effizient zu lösen. SMT-Solver wie Z3 bieten durch ihre Fähigkeit zur präzisen und schnellen Verarbeitung logischer Ausdrücke eine wertvolle Unterstützung bei der Verifikation und Validierung. Ein prominentes Beispiel für die Integration eines solchen Solvers ist die Software ProB. Der Animator, Constraint-Solver und Model-Checker ProB nutzt den SMT-Solver Z3, um formale Modelle effizient zu analysieren und zu überprüfen. Dies macht die Software zu einer wichtigen Intsanz in der Welt der formalen Methoden, insbesondere im Kontext von Modellierungs- und Verifikationsaufgaben.

1.1 Motivation

Innerhalb von ProB birgt der Einsatz des Z3-Solvers jedoch auch Herausforderungen, die die Effizienz und Zuverlässigkeit der Anwendung beeinträchtigen können. Ein bekanntes Problem besteht in dem sporadischen Auftreten von Speicherlecks und Segmentation Faults, die sowohl die Stabilität als auch die Nutzbarkeit von ProB's Z3-Interface negativ beeinflussen. Diese technischen Mängel erschweren nicht nur die Durchführung formaler Verifikationen, sondern können auch zu einer zeitraubenden Verwendung der Z3-Solver Komponente sowie Unterbrechung von Arbeitsprozessen führen.

Ein weiterer Mangel liegt in der aktuellen sequentiellen Lösung mehrerer Prädikate. Dieser Ansatz, bei dem die Prädikate nacheinander gelöst werden ist in seiner Natur ressourcenintensiv und zeitaufwändig. Angesichts der steigenden Komplexität formaler Modelle und der wachsenden Nachfrage nach schnellerer Verifikation wird die Limitierung durch die sequentielle Verarbeitung immer offensichtlicher. Eine Parallelisierung der Lösung von Prädikaten könnte hier erhebliche Leistungsverbesserungen bringen, indem moderne Mehrkernarchitekturen effizienter ausgenutzt werden, um den Anforderungen der Nutzer und der immer komplexer werdenden Modelle gerecht zu werden.

Die Kombination dieser Herausforderungen (sporadische technische Instabilitäten und

[1] Satisfiability Modulo Theories

begrenzte Effizienz durch sequentielle Verarbeitung) macht es notwendig, alternative Ansätze oder Verbesserungen für die Integration des Z3-Solvers in ProB zu erforschen und bildet die Grundlage und Motivation für die vorliegende Arbeit.

1.2 Ziele

Das Hauptziel dieser Arbeit ist es, die Integration des Z3-Solvers in ProB zu verbessern, indem die bestehende Vorgehensweise, die Prädikate direkt im Z3-Interface von ProB zu lösen, verworfen wird. Stattdessen wird eine neue Architektur vorgeschlagen und implementiert, welche eine vollständige Entkopplung der Z3-Solver-Komponente von ProB vorsieht. Hierzu wird also der Z3-Solver in einen eigenenständigen, seperaten Prozess ausgelagert, wodurch ein System eingeführt wird, bei dem ProB und der Z3-Solver als zwei unabhängige Prozesse agieren, die über eine Kommunikationsschnittstelle miteinander verbunden sind. Prädikate werden hierdurch innerhalb des Z3-Interfaces an den Z3-Solver gesendet, wo diese gelöst und zurückgeschickt werden. Diese geplante Architekturänderung ist in der folgenden Abbildung 1 visualisiert.

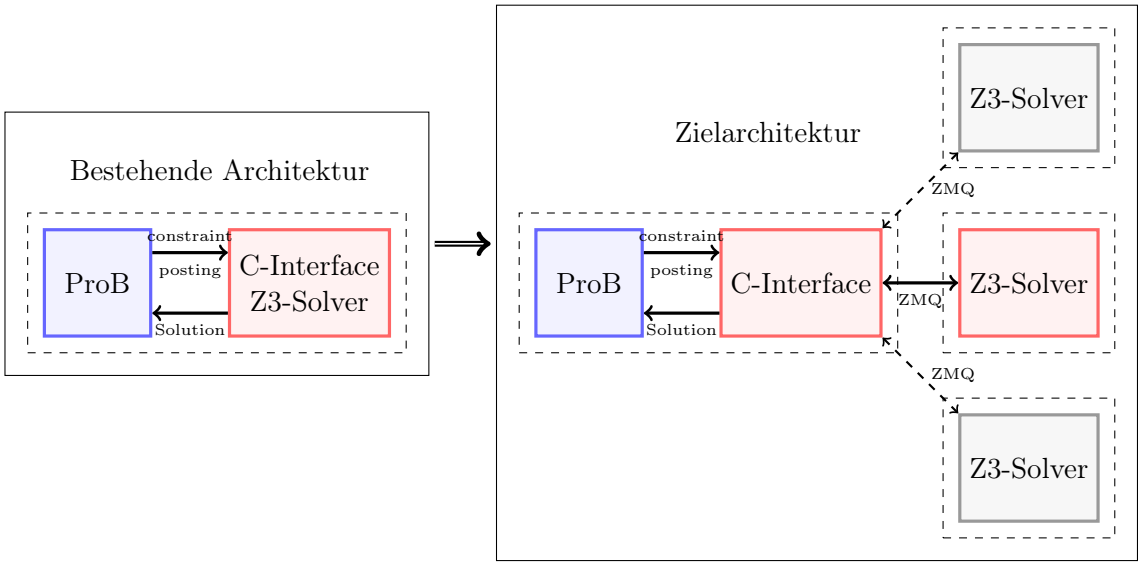

Abbildung 1: Die geplante Architekturänderung (Die Komponenten-Entkopplung ist in Rot markiert. Die gestrichelten Boxen zeigen die verschiedenen Prozesse an.)

Diese Arbeit wird einerseits mit dem Interesse der Erweiterbarkeit vollrichtet, sodass zukünftig die Option besteht, gegebenenfalls mehrere Instanzen des Z3-Prozesses zu starten und das Lösen der Prädikate zu parallelisieren. Andererseits dient die Entkopplung selbst bereits zur Verbesserung der Stabilität und Zuverlässigkeit von ProB, da bei eventuellen Fehlern im Z3-Solver-Prozess dieser unabhängig von ProB neu gestartet werden kann, was zu einem robusteren Gesamtsystem führt.

Die genauen Technologien und Konzepte, die hierfür zum Einsatz kommen und Relevanz zeigen, sowie ihre Funktionsweise und Vorteile werden im folgenden Kapitel detailliert erläutert. Daraufhin wird die Planung und Implementierung der neuen Architektur beschrieben, und die Leistungsfähigkeit der vorgenommenen Entkopplung anhand von Benchmarks und Tests hinsichtlich dem Vergleich zur vorherigen Systemstruktur evaluiert. Zuletzt wird auf zukünftige Erweiterungen und Verbesserungen eingegangen, und eine abschließende Konklusion genannt.

2 Grundlagen

Zur Förderung eines einheitlichen Verständnisses werden in diesem Abschnitt zunächst die erforderlichen Hintergrundinformationen illustriert. Im Folgenden werden die drei zentralen Konzepte behandelt, die für das Verständnis dieser Arbeit von Bedeutung sind: ProB, Z3 und ZeroMQ.

2.1 ProB

Die B-Methode, entwickelt von J.-R. Abrial [Abr96], ist eine formale Methode zur Entwicklung von Softwaresystemen, die auf der Idee der abstrakten Maschinen basiert. Mit abstrakten Maschinen lassen sich Zustände und deren Veränderungen mithilfe mathematischer Konzepte wie Mengen, Relationen und Funktionen modellieren [LB03]. Durch sogenannte Verfeinerungen wird schrittweise von einer abstrakten Beschreibung zu einer konkreten Implementierung übergegangen. Dabei stellt die Methode sicher, dass Invarianten stets eingehalten werden um die Korrektheit des Systems zu garantieren.

Die an der HHU am Lehrstuhl der Softwaretechnik und Programmiersprachen entwickelte Software ProB [LB03] ist ein Validierungs-Toolset für Modelle der B-Methode. Als solcher unterstützt ProB mitunter die Modellierung, Animation und Verifikation von B-Modellen, indem Funktionalitäten wie Consistency Checking und Constraint Solving bereitgestellt werden.

Der Animator in ProB ermöglicht es formale Spezifikationen zu visualisieren und zu animieren. Nutzer können durch die Simulation in Echtzeit einen Einblick in die Zustandsübergänge einer Maschine erhalten und schrittweise die Veränderungen nachvollziehen. Der aktuelle Zustand der Maschine wird dabei in einer grafischen Benutzeroberfläche dargestellt.

Ein weiterer Kernbestandteil von ProB ist das Consistency Checking, welches in zwei Ansätzen realisiert wird: Temporal Model Checking und Constraint-Based Checking.

Beim Temporal Model Checking wird versucht, eine Sequenz von Operationen zu finden, die, ausgehend von einem Anfangszustand, zu einer Verletzung der Invariante, oder einem anderen Fehler führt. Im Gegensatz dazu fokussiert sich das Constraint-based Checking auf die Suche nach einem Zustand des Systems, der die Invariante noch erfüllt. Von dort aus wird geprüft, ob es eine einzelne Operation gibt, welche die Invariante verletzt oder einen anderen Fehler erzeugt.

Während das Model Checking eine umfassende Exploration aller Zustände ermöglicht, ist das Constraint-based Checking spezifischer, da es nur auf Fehler bei einzelnen Operationen fokussiert ist. Zusammen lassen sich so vollständige Fehler und problematische Operationen identifizieren.

Beide Ansätze bieten wertvolle Instrumente für die Konsistenzprüfung von B-Modellen,

und sind in der Lage die Verletzung von Invarianten und daraus folgenden Bedingungen sowie die Abwesenheit von Deadlocks und das Erreichen von spezifizierten Zielprädikaten zu überprüfen [LB08].

Zuletzt bietet ProB auch eine Constraint-Solving-Funktionalität, die es ermöglicht, unter Berücksichtigung von gegebenen Constraints (Einschränkungen) Lösungen für spezifische Prädikate zu finden. Derartige Einschränkungen oder Bedingungen können in Form von logischen Ausdrücken oder Gleichungen gegeben sein, die es zu erfüllen gilt. Ein Constraint-Solver ist ein Algorithmus oder System, welches darauf abzielt unter Berücksichtigung eben jener Beindungen eine Belegung aller Variablen zu finden, die die gegebenen Prädikate erfüllt, und somit ein Problem auf dessen Erfüllbarkeit zu prüfen. ProB implementiert hierfür verschiedene Constraint-Solving-Strategien, die auf unterschiedlichen Algorithmen basieren und es ermöglichen, Prädikate effizient zu lösen. Einerseits wird CLP(FD)[2] verwendet, um auf endlichen Domänen beispielsweise Gleichheits- und Ungleichheitsbedingungen, sowie arithmetische Relationen zu lösen. Ein weiterer Ansatz ist die Integration des SAT-basierten Kodkod, einem effizienten Constraint-Solver für die Prädikatenlogik erster Ordnung mit Relationen, transitiven Hüllen, Bit-Vektor-Arithmetik und partiellen Modellen [TJ07]. Zuletzt wird auch der SMT-Solver Z3 in ProB integriert, um komplexere Prädikate zu lösen, die über simple boolsche und arithmetische Ausdrücke hinausgehen.

ProB ist im Kern in SICStus Prolog [CWA+88] implementiert, bietet jedoch verschiedene Programmerweiterungen, welche zumeist in C oder C++ geschrieben sind. Einer dieser Erweiterungen ist das Z3-Interface, welches die Integration des Z3-Solvers in ProB ermöglicht.

2.2 Z3 Solver

[BN24] [MB08]

2.3 ZeroMQ

[Hin13] [S+15]

[2] Constraint Logic Programming over Finite Domains

3 Architekturänderung

3.1 Planung

3.1.1 Prolog Datentypen

- atoms string - integers longs (laut dokumentation bis version blabla) - floats doubles - typerefs problem, weil kann alles sein

3.1.2 Struktur der Nachrichten

- function identifier - status identifier - message

3.1.3 Server Struktur

long damn switch case threading

3.2 Implementierung

3.2.1 Interfacefunktionen

porting of all 53 interface function

3.2.2 Hilfsfunktionen

inbesondere mk_type statemachines

3.2.3 Optimierungen

loops 2 von 4

manchmal $ctx_data- > blabla$ unnötige $ctx - data$

osstream to string

DRY with e.g. predicate in ||

3.2.4 Serveranbindung

server als subprozess starting as needed

3.2.5 Logging

via sys argv stdout not captureable in sicstus prolog

4 Exceptions

very important. need to be handled properly. need to work with server rep req structure what if multiple errors? need to notify other process

if function that can have an exception (even if handles eaxception) has an exception, return value will be invalid or empty. then the next code segment has exception because of that invalid return value, it is either not cought because not excpected and cant catch, or it is caught and we have 2 exceptions confusingly.

function a calls function b and c with b return. if b has exception and handles it itself, then it will, because sicstus keeps running until return of interface function, return invalid value, and keep running in a. maybe b does not expect an invalid input value and is uncaught, then segfault. or b now also has exception and then we have 2 in sicstus (confusing). additionally now server and prob have to keep req-rep ping-pong system. double exception feed back cannot work, because when ProBreceive exception

4.1 Kontrollfluss

dependency graph of all functions either throw or catch exceptions

5 Build Prozess

makefile build standalone executable etc...

6 Zusätzliche Ergebnisse

6.1 Softlock

endless loop fixed by interrupting on every reset

6.2 Versionsinkompatibilität

makefile hell glibc (OS) incompatible with z3lib.so -> darwin12

7 Leistungsbewertung

Nach Abschluss der durchgeführten Architekturänderung ist es elementar, die Auswirkungen auf die Laufzeitperformance zu bewerten. Da sich die grundlegende Funktionsweise des Z3-Solvers in der neuen Architektur nicht geändert hat, ist es zu erwarten, dass die Laufzeitperformance der ProB-Systemerweiterung durch die Einführung der ZeroMQ-Kommunikation negativ beeinflusst wird. Zusätzlich zu den Aufrufen des Z3-Interfaces müssen zur Lösung eines einzelnen Prädikates nun mehrere Anfragen und Antworten über den Socket serialisiert werden. Diese zusätzliche Kommunikation führt zu einem Performance-Overhead, welcher quantifiziert und evaluiert werden muss. Ebenfalls besteht ein Interesse zum Vergleich verschiedener ZeroMQ-Protokolle und deren Auswirkungen auf die Performance. Hierbei ist zu erwarten, dass das Inter-Process-Communication (IPC) Protokoll schneller ist als das Transmission-Control-Protocol (TCP) Protokoll, da es auf dem gleichen Rechner arbeitet und keine Netzwerkkommunikation benötigt, sondern das Dateisystem verwendet. Im nachfolgenden Abschnitt werden die Methodik der Leistungsbewertung, die erzielten Ergebnisse und ihre Interpretation detailliert beschrieben.

7.1 Performance-Overhead

Um eine Bewertung des Performance-Overheads zu ermöglichen, müssen zunächst empirische Daten erhoben werden. Hierzu werden die Tests zur Verifikation der Funktionsweise des Z3-Interfaces umfunktioniert, um die Laufzeit der einzelnen Anfragen zu messen. Im Code des Z3-Interfaces wird ein Zeitstempel bei Beginn und Ende des Lösungsvorgangs eines Prädikates gesetzt, dessen Differenz berechnet und gespeichert. Insgesamt stehen 53 Tests zur Verfügung, die in der Testumgebung des Z3-Solvers ausgeführt werden können. Diese Tests umfassen insgesamt 679 Prädikate, welche eine ausreichende Grundgesamtheit zur Bewertung der Performance bieten. Ebenfalls wird die Anzahl der Anfragen und Antworten, die über das Netzwerk gesendet werden, gemessen. Ein kleiner Auszug dieser Messdaten sind in Tabelle 1 dargestellt.

Tabelle 1: Auszug der Daten der Performance-Messung.

TestID	QueryID	Old	New(IPC)	New(TCP)	RequestCounter
⋮	⋮	⋮	⋮	⋮	⋮
1510	1	114815014	283392117	113503997	191
1510	2	52048678	59273375	44489012	166
1510	3	30853103	24983820	25212332	286
1511	1	69240686	199325313	61823314	21
1511	2	61404523	62220124	48522297	20
1513	1	80829324	202694845	75877790	25
⋮	⋮	⋮	⋮	⋮	⋮

Die Zeitmessungen sind in Nanosekunden (ns) angegeben und zeigen die Laufzeit der Anfragen in den verschiedenen Konfigurationen. Innerhalb der gegebenen Größenordnung sind die Werte ebenfalls in Millisekunden (ms) zu interpretieren und damit ausschlaggebend für die Performance. Innerhalb der eigentlichen Daten wurden die Messungen mehrfach unabhängig voneinander wiederholt, um eine statistische Aussagekraft zu gewährleisten. Da dennoch von Messfehlern und Schwankungen auszugehen ist, werden die Daten in einem statistischen Kontext betrachtet. Es wird angenommen, dass die Messfehler normalverteilt sind.

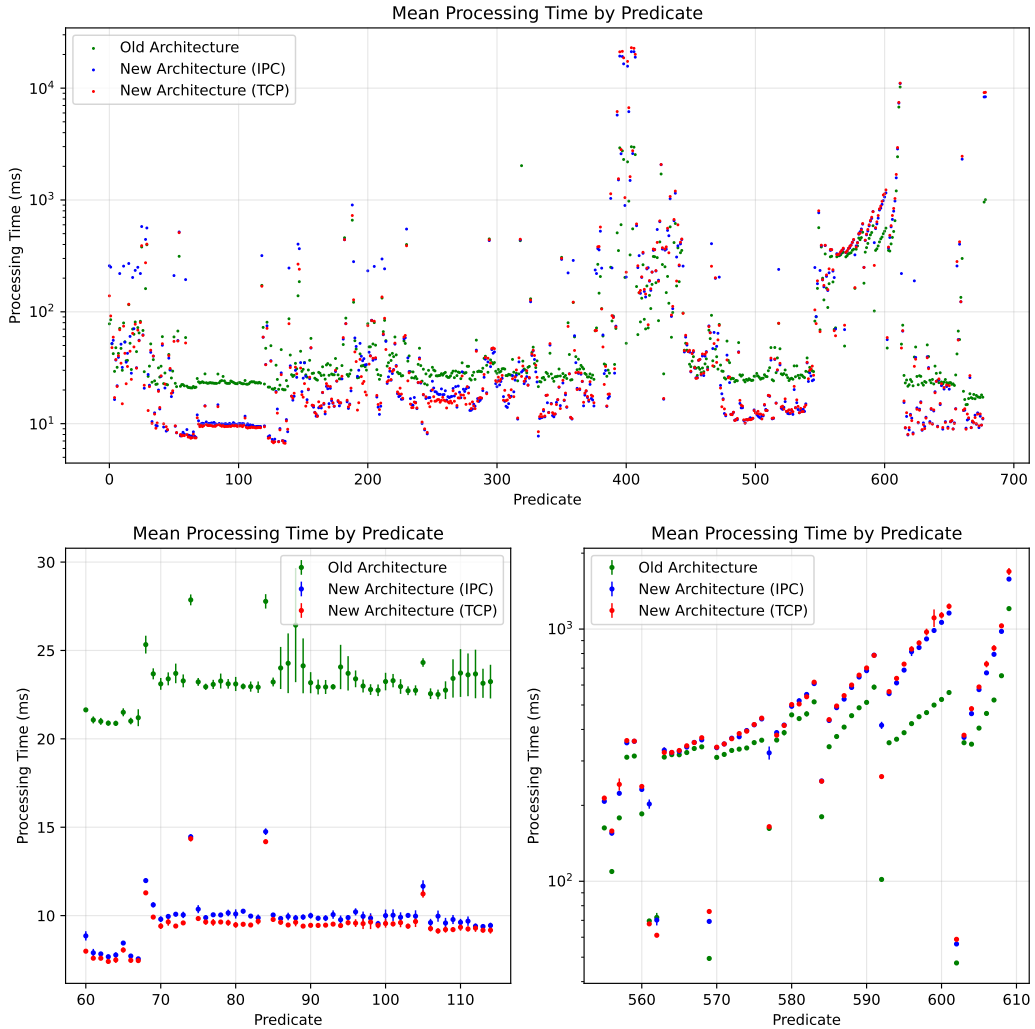

Abbildung 2: Durchschnittliche Laufzeiten der Anfragen in den verschiedenen Architekturen.

Um sich zunächst einen Überblick über die Rohdaten zu verschaffen, werden in Abbildung 2 die durchschnittlichen Laufzeiten in den verschiedenen Konfigurationen dargestellt. Der obere Subgraph zeigt die durchschnittlichen Laufzeiten aller Anfragen in sowohl der alten

als auch der neuen Architektur, wobei die neue Architektur in das IPC-Protokoll und TCP-Protokoll unterteilt ist. Wider Erwarten zeigt sich, dass die durchschnittliche Laufzeit in der neuen Architektur bei vielen Prädikaten geringer ausfällt als in der alten Architektur. Insgesamt sind von den 679 Prädikaten nur 172 Prädikate in der alten Architektur schneller gelöst worden. 265 Prädikate wurden in der IPC-Konfiguration und 242 Prädikate in der TCP-Konfiguration am schnellsten gelöst.

Die unteren beiden Subgraphen stellen interessante Bereiche der Rohdaten erneut dar und zeigen zusätzlich die Standardfehler der Mittelwerte in Form der Fehlerbalken. Diese geben an, wie sehr die Mittelwerte der Laufzeiten von den tatsächlichen Mittelwerten der Grundgesamtheit abweichen. Kalkuliert wird dieser Standardfehler mittels der folgenden Formel:

$$\sigma_{\overline{x}} = \frac{\sigma}{\sqrt{n}} \tag{1}$$

Hierbei ist $\sigma_{\overline{x}}$ der Standardfehler des Mittelwertes, σ die Standardabweichung der Grundgesamtheit und n die Anzahl der Messungen. Somit werden ausschließlich statistische Fehler betrachtet und systematische Fehler nicht berücksichtigt.

So zeigt der untere linke Subgraph die Rohdaten im Bereich der Prädikate 60 bis 115 erneut, wobei klar zu erkennen ist, dass hier die neue Server-Architektur konstant schneller ist. Dieser Trend ist über die gesamte Messung hin unterhalb einer gewissen Grenze zu beobachten. Ebenfalls ersichtlich ist der nur minimal ausfallende Unterschied zwischen IPC und TCP.

Auf der anderen Seite zeigt der untere rechte Subgraph die Rohdaten im Bereich der Prädikate 555 bis 610. Hier haben die Laufzeit Messungen einen vergleichsweise hohen Wert und es ist zu erkennen, dass die alte Architektur in diesem Bereich schneller ist.

Entsprechend wurde durch die Protokollkommunikation ein Performance-Overhead eingeführt, der sich in den Rohdaten dahingehend widerspiegelt, dass Prädikate, welche bereits eine hohe Laufzeit aufgewiesen haben, nun in der neuen Architektur noch langsamer gelöst werden. Neben diesem Overhead wurde jedoch eine signifikante Laufzeitverbesserung erworben, die bis zu einer gewissen Gesamtlaufzeitsgrenze den Overhead nicht nur annulliert, sondern überkompensiert. Insgesamt scheint die neu eingeführte Server-Architektur hinsichtlich der Laufzeitperformance um einen gewissen Faktor effizienter geworden zu sein.

Ebenfalls ist zu erkennen, dass der Unterschied zwischen IPC und TCP nur minimal ist und keine signifikanten Unterschiede aufweist. Innerhalb der 679 Testprädikaten ist IPC in 365 (53.76%) Fällen schneller als TCP und entsprechend TCP in 314 (46.24%) Fällen schneller als IPC. Da zur TCP-Kommunikation die Adresse $tcp://127.0.0.1$ verwendet wurde, ist eine mögliche Erklärung hierfür, dass die Kommunikation über das Loopback-Interface einen Großteil des Netzwerk-Stacks umgeht und vom Betriebssystem speziell optimiert wird. In der weiteren Analyse wird daher nur das IPC-Protokoll betrachtet.

Um den Overhead der neuen Architektur zu quantifizieren, wird die Differenz der durchschnittlichen Laufzeiten der neuen und alten Architektur berechnet und gegen die Anzahl der ZeroMQ-Anfragen analysiert. Der induzierte Overhead wird also wie folgt definiert:

$$InduzierterOverhead = \overline{IPCLaufzeit} - \overline{AlteLaufzeit} \tag{2}$$

Der oberste Subgraph in Abbildung 3 zeigt den induzierten Overhead der neuen Architektur in Abhängigkeit der Anzahl der ZeroMQ-Anfragen auf einer logarithmischen Skala. Im groben Verlauf ist zu erkennen, dass der Overhead mit steigender Anzahl der Anfragen annähernd linear zunimmt. Zusätzlich scheint jedoch eine ausgeprägte Systematik vorzuliegen, welche sich inhaltlich durch die horizontale Verzerrung der Daten ausprägt. Einige wenige Datenpunkte weichen stark von jeglicher Systematik ab und sind als Ausreißer zu klassifizieren.

Im zweiten Subgraphen wird die lineare Abhängigkeit des Overheads von der Anzahl der Anfragen untersucht, indem der Zusammenhang linear gefittet wird. Dies geschieht unter Berücksichtigung des kombinierten Standardfehlers der Mittelwerte der alten und neuen IPC-Architektur

$$\sigma_{\overline{comb}} = \sqrt{\sigma^2_{\overline{IPC}} + \sigma^2_{\overline{Old}}} \quad , \tag{3}$$

welche ebenfalls in den Fehlerbalken dargestellt wird. Das Ergebnis des linearen Fits zeigt eine Steigung von $0.03ms$ pro Anfrage, was bedeutet, dass der Overhead pro Anfrage um $0.03ms$ steigt. Zusätzlich gibt es einen y-Achsenabschnitt von $-13.13ms$, was bedeutet, dass der Overhead bei 0 Anfragen $-13.13ms$ beträgt. Dieser Wert beschreibt die zuvor erfasste Laufzeitverbesserung der neuen Architektur. Insgesamt ergibt sich die folgende lineare Funktion zur Beschreibung des Overheads:

$$Overhead = 0.03ms \cdot Anfragenanzahl - 13.13ms \tag{4}$$

Durch das Gleichsetzten der Overhead-Funktion mit 0 ergibt sich eine Anzahl an Anfragen von etwa 438, bis der Overhead nicht länger von der Laufzeitverbesserung kompensiert werden kann.

Eine separate Messreihe wurde durchgeführt, um die durchschnittliche Laufzeit einer einzelnen Anfrage an den Z3-Server zu ermitteln. Hierbei wurde die durchschnittliche Laufzeit von $0.0054ms$ bestimmt. Da für jede Anfrage zwei Nachrichten (Anfrage und Antwort) über den Socket gesendet werden, ergibt sich eine Laufzeit von $0.0216ms$ pro Anfrage unter der Annahme, dass das Erhalten einer ZeroMQ-Nachricht die gleiche Laufzeit aufweist wie das Senden einer Nachricht. Die Differenz von $0.0084ms$ zur ermittelten Steigung des linearen Fits ist beispielsweise auf die Größe der Nachrichten zurückzuführen, welche in der kontrollierten Messreihe einer einzelnen Anfrage kleiner ausfällt als in den 52

Testszenarien. Zudem sind Messwerte dieser Größenordnung zunehmend ungenau, weshalb sie annehmbar plausibel bezüglich der Steigung des linearen Fits sind.

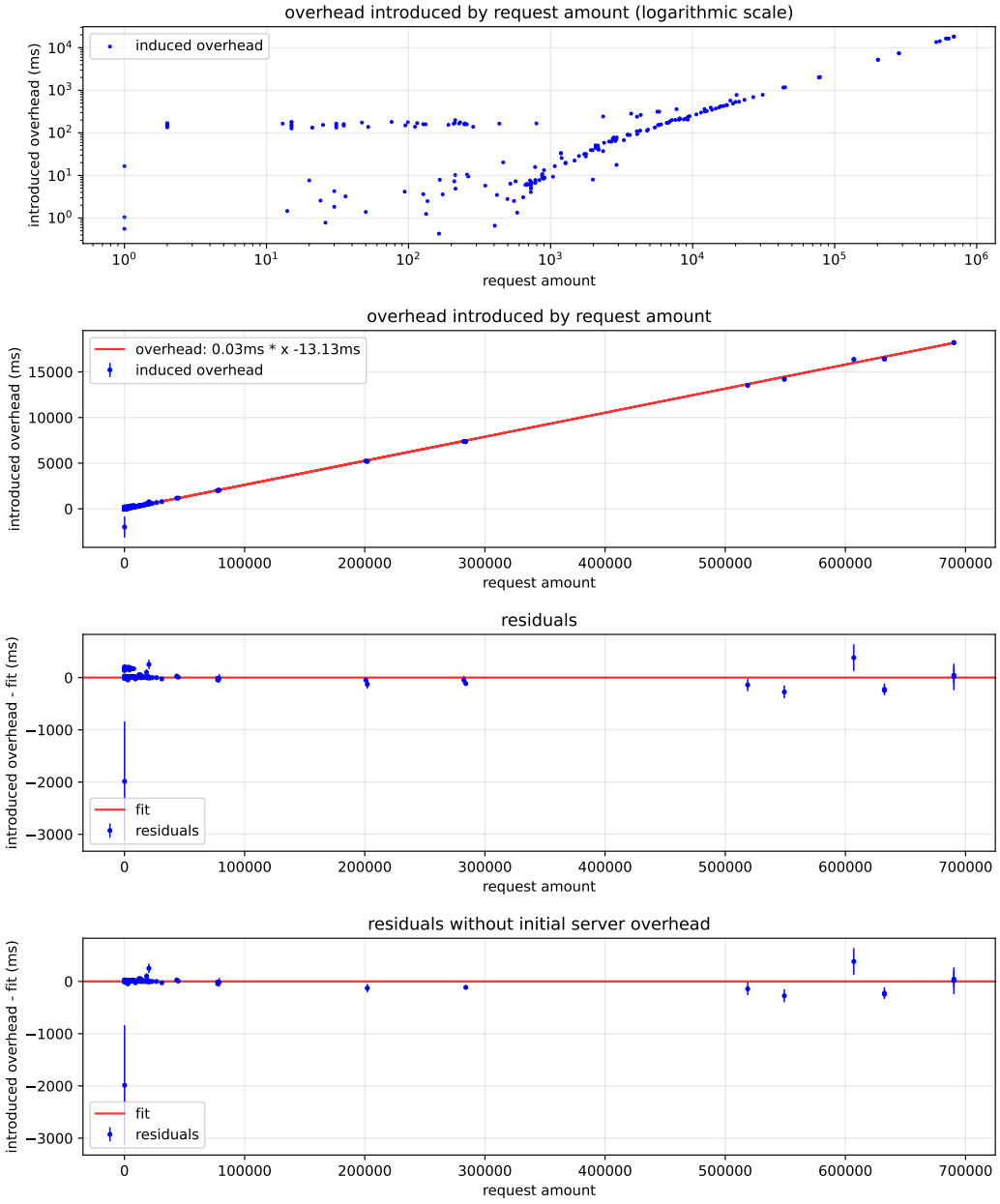

Abbildung 3: Induzierter Overhead der neuen Server-Architektur durch das Serialisieren auf den Socket.

Der lineare Fit zeigt augenscheinlich eine gute Übereinstimmung mit den Daten, welche in dem dritten Subgraphen aus Abbildung 3 verifiziert wird. Hierbei wird die Differenz des induzierten Overheads und des linearen Fits gegen die Anzahl der Anfragen dargestellt. Es ergibt sich ein Residuenplot. In diesem Plot ist die Systematik des obersten Subgraphen erneut zu erkennen. Sie bildet ein kleines Cluster oberhalb der 0-Linie, bei sehr kleinen Anfragenzahlen. Es stellt sich heraus, dass innerhalb dieses Datenclusters alle Datenpunkte die QueryID 1 aufweisen. Somit lässt sich die Abweichung durch das Initialisieren des Sockets, der Verbindung des Z3-Servers und dessen initialen Overheads zum Aufsetzten der Z3-Konfigurationen erklären. Innerhalb einer TestID wird der Z3-Server nur einmalig initialisiert und folgende Prädikate desselben Tests nutzen denselben Z3-Server, indem sie die Konfiguration zurücksetzten und nicht grundlegend neu initialisieren.

Zur Überprüfung stellt der letzte Subgraph erneut den Residuenplot dar, jedoch ohne diejenigen Datenpunkte mit QueryID 1. Hierbei ist zu erkennen, dass die Systematik des vorherigen Graphen verschwindet und der lineare Fit eine gute Übereinstimmung mit den Daten aufweist. Somit ist gezeigt, dass der Overhead der neuen Architektur durch das Serialisieren auf den Socket annähernd linear mit der Anzahl der Anfragen zunimmt und keine weiteren relevanten Systematiken besitzt. Eine Voraussage des Overheads ist mithilfe des linearen Fits möglich. Es ist zu beachten, dass die absoluten Werte des Overheads nur in der gegebenen Testumgebung gültig sind und nicht auf andere Umgebungen übertragen werden können.

Ein einzelner Datenpunkt scheint eine unwahrscheinlich hohe Laufzeitverbesserung von durchschnittlich 2 Sekunden aufzuweisen, zeichnet jedoch einen hohen Fehlerbalken auf. In Tabelle 2 ist dieser Datenpunkt dargestellt.

Tabelle 2: Ausschnitt der gesammelten Messwerte eines Ausreißers.

TestID-QueryID	Old_1	Old_2	Old_3	Old_4	**RequestCounter**
2122-90	36	4033	3993	43	119

Die verschiedenen Messwerte der alten Architektur in Millisekunden zeigen zwei starke Ausreißer bei der zweiten und dritten Messung. Diese Varianz ist womöglich auf die in Abschnitt 6.1 beschriebene Problematik zurückzuführen. In jedem Fall ist dieser Datenpunkt als individueller Ausreißer zu betrachten und weist keine besondere statistische Relevanz auf.

Die ermittelte Performance-Verbesserung wird an dieser Stelle nicht weiter analysiert, da sie nicht Gegenstand dieser Arbeit ist. Ein möglicher Grund für die Verbesserung könnte die Kompilierung sein, die durch die neue Architektur ermöglicht wird, da der Z3-Solver als eigenständiger Prozess womöglich besser vom Compiler optimiert werden kann. Als eigenständiger Prozess ist es ebenfalls möglich, dass der runtime linker die Z3-Bibliothek optimierter laden und ausführen kann. Zusätzlich könnten die in Abschnitt 3.2.3 beschriebenen Optimierungen eine Rolle spielen.

8 Zukünftige Arbeiten

8.1 Weitere Analyse und Optimierungen

analyse performance improvement

8.2 Deinit Hook

deinit für dangling zocket und prozess (cleaner)

8.3 Parallelisierung

threading in ProB (eigentlicher Sinn der Arbeit um zu parallelisieren)

9 Konklusion

stuff

Literatur

[Abr96] Abrial, Jean-Raymond: *The B-Book: Assigning Programs to Meanings.* New York, NY, USA : Cambridge University Press, 1996

[BN24] Bjørner, Nikolaj ; Nachmanson, Lev: Arithmetic Solving in Z3. In: Gurfinkel, Arie (Hrsg.) ; Ganesh, Vijay (Hrsg.): *Computer Aided Verification.* Cham : Springer Nature Switzerland, 2024. – ISBN 978–3–031–65627–9, S. 26–41

[CWA+88] Carlsson, Mats ; Widen, Johan ; Andersson, Johan ; Andersson, Stefan ; Boortz, Kent ; Nilsson, Hans ; Sjöland, Thomas: *SICStus Prolog User's Manual.* Kista, Sweden : Swedish Institute of Computer Science, 1988

[Hin13] Hintjens, P: *ZeroMQ: Messaging for Many Applications.* O'Reilly Media, 2013

[LB03] Leuschel, Michael ; Butler, Michael: ProB: A model checker for B. In: *FME 2003: Formal Methods: International Symposium of Formal Methods Europe, Pisa, Italy, September 8-14, 2003. Proceedings* Springer, 2003, S. 855–874

[LB08] Leuschel, Michael ; Butler, Michael: ProB: an automated analysis toolset for the B method. In: *International Journal on Software Tools for Technology Transfer* 10 (2008), S. 185–203

[MB08] Moura, Leonardo de ; Bjørner, Nikolaj: Z3: An Efficient SMT Solver. In: Ramakrishnan, C. R. (Hrsg.) ; Rehof, Jakob (Hrsg.): *Tools and Algorithms for the Construction and Analysis of Systems.* Berlin, Heidelberg : Springer Berlin Heidelberg, 2008. – ISBN 978–3–540–78800–3, S. 337–340

[S+15] Sústrik, Martin u. a.: ZeroMQ. In: *Introduction Amy Brown and Greg Wilson* (2015), S. 16

[TJ07] Torlak, Emina ; Jackson, Daniel: Kodkod: A relational model finder. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* Springer, 2007, S. 632–647