Hochschule Esslingen
University of Applied Sciences

Live-Analyse von IO-Daten in HPC-Anwendungen

im Studiengang Angewandte Informatik
der Fakultät Informationstechnik
Wintersemester 2023/2024

Nico Linder

Betreuer: Prof. Dr.-Ing. Rainer Keller, Philipp Köster

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

Im Bereich des High Performance Computing (HPC) zeichnet sich mit der Verfügbarkeit immer höherer Rechenleistung die Problematik ab, dass die zugrundeliegenden IO-Systeme bei Anwendungen, mit denen große Datenmengen verarbeitet werden, nicht mit der rasanten Entwicklung der Parallelisierung und Verarbeitungsgeschwindigkeit Schritt halten können. Dieser sogenannte IO-Bottleneck wurde bereits im Jahr 2014 als eines der Hauptprobleme für die Performance von datenintensiven HPC-Anwendungen im wissenschaftlichen Bereich bezeichnet und wird in Zukunft noch stärker von Bedeutung sein, da die Rechenleistung von HPC-Systemen deutlich stärker ansteigt, als die verfügbare IO-Bandbreite [3].

Während einige Arbeiten darauf abzielen, dieses Problem durch neuartige, speziell für Anwendungen mit hohem Datenaufkommen entwickelte IO-Mechanismen zu lösen, welche die zur Verfügung stehende IO-Bandbreite der HPC-Systeme selbst erhöhen sollen [3][7], könnte ein alternativer Ansatz darin bestehen, das IO-Verhalten von Programmen zu analysieren, um ineffiziente IO-Muster automatisiert zu erkennen und so Entwicklern zu ermöglichen, die bestehenden Ressourcen durch Optimierung der Anwendung besser auszunutzen.

1.1 Motivation

Rechenzeit auf HPC-Systemen ist kostenintensiv und sehr gefragt, weshalb Anwendungen auf diesen Systemen möglichst effizient ausgeführt werden sollen. Bei wissenschaftlichen Anwendungen, die große Datenmengen verarbeiten müssen, ist der limitierende Faktor hier in vielen Fällen nicht mehr die Rechengeschwindigkeit des HPC-Systems, sondern die begrenzte IO-Bandbreite und Latenz angebundener Dateisysteme wie zum Beispiel Lustre, wenn häufig Daten darauf geschrieben oder davon gelesen werden müssen. Während die Zugriffszeit auf Daten im Arbeitsspeicher des Clusters im Nanosekundenbereich liegt, bewegt sich die Zugriffszeit selbst bei schnellen SSD-Festplatten im Bereich von Millisekunden, was einer Verlangsamung um den Faktor 1000 entspricht. Um die Rechenleistung eines HPC-Systems optimal auszunutzen, sollte der File-IO von Programmen deshalb so effizient wie möglich gestaltet werden.

Zur Analyse von File-IO in HPC-Anwendungen wurde im Rahmen von Forschungsprojekten in vergangenen Semestern bereits das Tool *libiotrace* entwickelt. Dieses protokolliert, je nach Konfiguration, verschiedene Arten von Dateisystem-Zugriffen (POSIX-IO, MPI-IO) des überwachten Programms und speichert die entsprechenden Daten wahlweise in einer parallellaufenden Zeitreihendatenbank oder in einer Log-Datei. Als Zeitreihendatenbank wird InfluxDB genutzt. Weiterhin existiert bereits ein Tool, um die Log-Datei nach dem Ende der Ausführung des überwachten Programms auszuwerten und verschiedene Statistiken über die aufgezeichneten IO-Zugriffe grafisch darzustellen, sowie eine Grafana-Instanz, mit der die in der InfluxDB gespeicherten Daten visualisiert werden können.

Anhand der IO-Daten, die mit Hilfe der libiotrace aufgezeichnet werden, sollen nun ineffiziente IO-Muster, die das Programm ausbremsen, erkannt und dem Benutzer gemeldet werden. Idealerweise soll dies soweit möglich schon zur Laufzeit des Programms geschehen, da die betreffenden Anwendungen häufig sehr lange Laufzeiten haben und „schlechte“ IO-Muster auf diese Weise schnell erkannt und ggf. behoben werden können, ohne jedes Mal die Ausführung des kompletten Programms abzuwarten.

1.2 Ziele

In diesem Forschungsprojekt soll das bestehende Framework zur Verfolgung und Visualisierung von IO-Daten erweitert werden, um ineffiziente IO-Muster, welche die Laufzeit von Programmen negativ beeinflussen, zu erkennen. Dies soll möglichst parallel zur Ausführung des untersuchten Programmes geschehen, sodass zur Messung des File-IO nicht immer das komplette Programm durchlaufen werden muss und schnell auf problematisches IO-Verhalten reagiert werden kann.

Dazu soll zunächst untersucht werden, wie „schlechtes“ IO-Verhalten erkannt werden kann. Zu diesem Zweck sollen auch existierende Werkzeuge und Ansätze zur IO-Klassifikation betrachtet werden. Die Weiterverarbeitung bzw. Auswertung der von der libiotrace gesammelten Daten soll primär mit der von InfluxData entwickelten Skriptsprache Flux direkt beim Abfragen aus der Datenbank geschehen, aber auch eine Auswertung bzw. Aufbereitung der Daten außerhalb der InfluxDB wäre denkbar, sollte dies erforderlich werden.

Die Ergebnisse der Auswertung sollen auf einem Grafana-Dashboard visualisiert werden. Hierbei ist es letztendlich das Ziel, dem Nutzer nicht nur reine IO-Statistiken anzuzeigen, sondern auch konkrete Hinweise auf erkannte schlechte IO-Muster sowie Vorschläge, um die Performance zu verbessern.

2 Grundlagen und Stand der Technik

In diesem Kapitel werden die zur File-IO-Analyse verwendeten Werkzeuge sowie Grundlagen kurz beschrieben. Außerdem wird ein Überblick gegeben, welche Werkzeuge und Ansätze zur IO-Charakterisierung bereits am Markt existieren. Im Rahmen der Recherche nach existierenden Lösungen wurde sowohl nach Software-Lösungen, die IO-Tracing und -Charakterisierung anbieten gesucht, als auch nach Publikationen, die Ansätze in diesem Gebiet beschreiben. Zur Suche nach wissenschaftlichen Publikationen wurde Google Scholar [4] sowie die ACM Digital Library [1] verwendet.

2.1 InfluxDB

InfluxDB ist ein Open Source-Datenbankmanagementsystem zur Speicherung von Zeitreihendaten. Es ist besonders gut geeignet für die Speicherung einer großen Anzahl von Messdaten, die über einen bestimmten Zeitraum anfallen. Zur Abfrage von Daten gibt es zwei Möglichkeiten: Die SQL-ähnliche Sprache InfluxQL und die von Grund auf neu entwickelte Skriptsprache Flux. Beide Sprachen ermöglichen es, Daten abzufragen, zu analysieren und zu transformieren [5]. InfluxDB bietet sowohl integrierte Werkzeuge zur Visualisierung von Daten in verschiedenen Grafiken und Diagrammen, als auch die Möglichkeit, für externe Visualisierungstools wie Grafana als Datenquelle zu fungieren [2] [5].

Die libiotrace protokolliert alle IO-Aufrufe des überwachten Programms in einer parallel zu dem Programm auf dem HPC-System laufenden InfluxDB-Instanz. Zu den protokollierten Daten gehören u.A. der genaue Funktionsaufruf (POSIX oder MPI), Dauer und Größe der IO-Operation sowie Dateinamen und Informationen zum Dateisystem. Mit diesen Daten soll die IO-Klassifizierung realisiert werden.

2.2 IO-Charakterisierung

Es existieren bereits eine Reihe von Werkzeugen und Ansätzen, mit denen die IO-Aktivitäten von Programmen auf HPC-Clustern nicht nur nachverfolgt und protokolliert, sondern auch analysiert werden können, um durch File-IO verursachte Performance-Probleme zu finden.

2.2.1 Darshan

Darshan ist ein IO-Profiling-Tool, das detaillierte Statistiken über die IO-Workloads von HPC-Anwendungen erstellt. Dazu werden, ähnlich wie bei der libiotrace, die Dateizugriffe des überwachten Programms abgefangen und Statistiken, Timing-Daten und weitere Informationen zur IO-Charakterisierung extrahiert, bevor der IO-Aufruf an die Systembibliotheken weitergegeben wird. Diese Daten werden anschließend komprimiert in einer Log-Datei zur späteren Auswertung gespeichert [9].

Die Auswertung geschieht entkoppelt von der Programmausführung auf einem anderen System mithilfe des Programms *darshan-util*. Dieses erstellt einen Bericht über das IO-Verhalten in Form einer PDF-Datei, in welchem die ermittelten Daten grafisch dargestellt werden. Abbildung 2.1 zeigt einen Auszug aus einem Beispiel-Bericht. Die ermittelten Daten umfassen unter anderem [6]:

- Den Anteil von Schreib- und Leseoperationen an der Gesamtlaufzeit des Programms (getrennt für POSIX und MPI)
- Gesamte Anzahl an IO-Zugriffen aller Prozesse (Read, Write, Open, usw.)
- Durchschnittliche Anzahl an IO-Zugriffen pro Prozess
- Größe der einzelnen Datei-Zugriffe sowie die am häufigsten auftreten Zugriffsgrößen
- Gesamte Anzahl an Dateien, auf die zugegriffen wurde
- Zeitraum vom ersten bis zum letzten lesenden bzw. schreibenden Zugriff auf Dateien

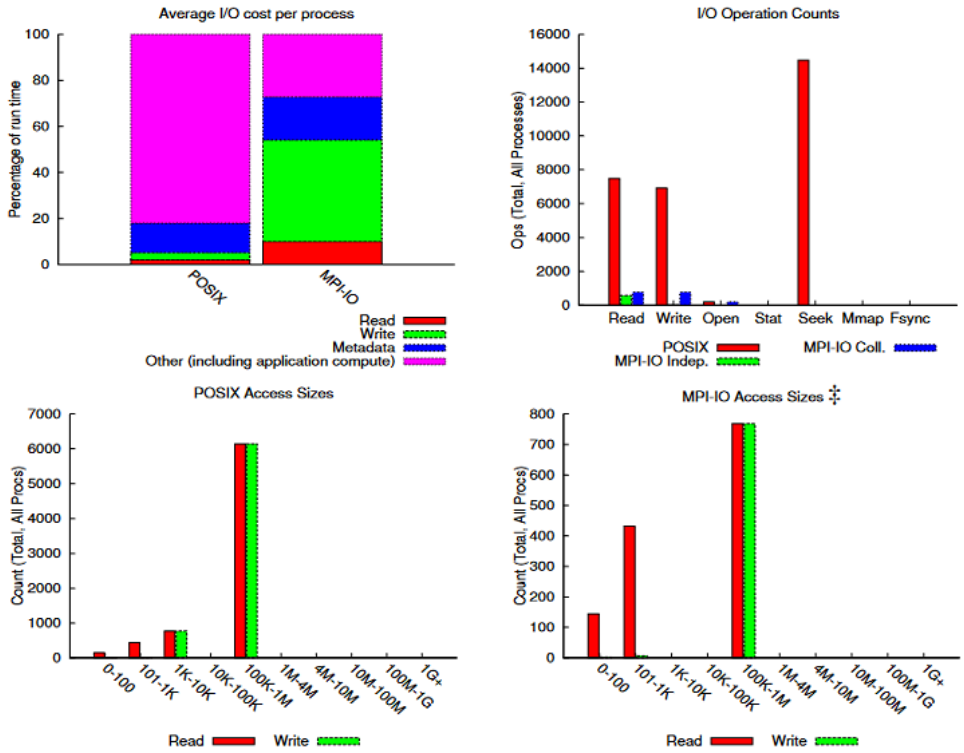

Abb. 2.1: Ausschnitt aus der Ausgabe von *darshan-util* [6]

2.2.2 ScalaIOTrace

ScalaIOTrace ist ein IO-Tracing-Tool der North Carolina State University, das auf ScalaTrace basiert und ebenfalls MPI- und POSIX-Systemaufrufe abfängt und protokolliert. Im Gegensatz zu vielen anderen Tools legt ScalaIOTrace großen Wert darauf, die entstehenden Tracing-Daten so stark wie möglich zu komprimieren, um auch für Anwendungen auf sehr großen Clustern skalierbar zu bleiben [10].

Die Analyse der gesammelten IO-Daten erfolgt bei ScalaIOTrace über sogenannte *replays*. Dabei werden die aufgezeichneten IO-Operationen (und optional auch MPI-Kommunikation) über eine replay engine noch einmal in der gleichen Ablaufreihenfolge abgespielt, wie es im analysierten Programm der Fall war. Alle IO-Zugriffe werden mit den gleichen Parametern wiederholt, es werden lediglich die geschriebenen bzw. gelesenen Daten durch Zufallsdaten ersetzt. Die Rechenzeit zwischen den IO-Aufrufen wird ebenfalls simuliert [10].

Auf diese Weise ermöglicht ScalaIOTrace, das IO-Verhalten des untersuchten Programms nach dessen einmaliger Ausführung beliebig oft wieder „abzuspielen“ und dabei zu analysieren. Dies kann vorteilhaft sein, wenn nicht im Voraus bekannt ist,

nach welchen problematischen IO-Mustern gesucht werden soll. Außerdem können so iterative Verbesserungen vorgenommen und getestet werden, ohne das Programm jedes Mal wieder erneut ausgeführt werden muss [10].

# nodes	I/O at 0	Coll. at 0	Block. at 0	NB at 0	Coll. Other	Block. Other	NB Other
2	1589	21247	129034	231714	21247	0	385350
4	1573	21257	179284	308952	21257	0	388838
8	1573	21277	210140	308952	21277	0	393393
16	1573	21317	1444912	386190	21317	0	447680
32	1573	21397	858648	386190	21397	0	451373
64	1573	12225	858648	386190	8575	0	382512
128	1573	21877	463372	386190	21877	0	441344
256	1573	22517	470288	386190	22517	0	426550
512	1573	23797	239932	386190	23797	0	424329
1024	1573	26357	240198	386190	26357	0	412485

Abb. 2.2: Von ScalaIOTrace erfasste Funktionsaufrufe des Parallel Ocean Program [10]

2.2.3 TAU Performance System

TAU ist ein weiteres Tool, mit dem IO-Aufrufe auf HPC-Systemen verfolgt und analysiert werden können. Für die Auswertung der Daten stellt Tau für jeden Prozessor grafisch das Verhältnis zwischen für Berechnungen und für IO aufgewendete Zeit dar. Für die IO-Phase kann außerdem dargestellt werden, welche Funktion (MPI write/read, POSIX write/read) welchen Anteil an der IO-Zeit hatte. Außerdem wird auch für jeden Prozessor die Größe der Read- und Write-Zugriffe sowie die Bandbreite berechnet [8].

Die Interpretation der gesammelten Daten bleibt bei TAU dem Nutzer überlassen, so muss beispielsweise erkannt werden, dass das allgemeine Verhältnis zwischen IO und Rechenzeit unpassend ist und dann anhand der weiteren Statistiken auf mögliche Ursachen geschlossen werden [8].

2.2.4 Fazit

Darshan nimmt von den betrachteten Werkzeugen die umfangreichste Auswertung vor und präsentiert diese in grafischer Form, was in ähnlicher Form auch für das Grafana-Dashboard der libiotrace umgesetzt werden könnte. Die verwendeten Metriken geben einen guten Ansatz für weitere Arbeit an der automatischen Erkennung von schlechtem IO, so könnte beispielsweise das Einhalten bestimmter Grenzwerte überwacht werden.

Die von ScalaIOTrace gesammelten Statistiken zu IO-Aufrufen sind deutlich weniger umfangreich als es beispielsweise bei Darshan der Fall ist, siehe Abbildung 2.2. Es scheint, als wolle ScalaIOTrace vor allem ein Framework für die effiziente Aufzeichnung von IO-Traces auf sehr großen Clustern bieten und die Implementierung der Auswertung größtenteils dem Anwender überlassen.

TAU bietet prinzipiell eine ähnliche Auswertung der Daten wie Darshan, jedoch weniger umfangreich. Interessant ist jedoch, dass hier die Leistung aller Prozessoren getrennt betrachtet und dargestellt werden, während bei Darshan meist ein Durchschnitt über alle Prozesse gebildet wird.

3 Umsetzung

Um die Klassifizierung und Erkennung von schlechten IO-Mustern zu realisieren, soll evaluiert werden, ob die dafür nötige Auswertung der von der libiotrace protokollierten IO-Aufrufe mithilfe von Flux-Queries direkt aus der InfluxDB heraus vorgenommen werden kann. Dazu sollen zunächst einige Statistiken über den File-IO der untersuchten Anwendung erzeugt werden, die auch im Grafana-Dashboard visualisiert werden können. Anschließend kann nach Ansätzen gesucht werden, wie anhand der verfügbaren Daten und Statistiken schlechtes IO-Verhalten erkannt werden kann.

3.1 Testdaten

Um Daten zum Abfragen in die Datenbank zu schreiben, müssen mit der libiotrace Programmausführungen protokolliert werden. Hierzu steht bereits eine Reihe von einfachen Tests-Scripts für POSIX- und MPI-IO zu Verfügung, allerdings erzeugen diese sehr vorhersehbare Muster, die sich nicht zur späteren Analyse auf schlechte IO-Muster eignen.

Eine Möglichkeit, bessere Testdaten zu erzeugen, war das Schreiben von eigenen, kurzen Programmen mit IO-Befehlen, die dann mit der libiotrace untersucht werden können. Listing A.1 zeigt ein einfaches Programm in der Sprache Elixir, das in einer Schleife Zeilen in eine Datei schreibt.

Hintergrund ist hier, dass Dateizugriffe in Elixir jeweils in einem eigenen Prozess und somit asynchron geschehen. Dies wäre eine gute Möglichkeit gewesen zu untersuchen, wie sich viele parallele, nicht optimierte Zugriffe auf eine Datei in den Ergebnissen des IO-Tracing auswirken und wie man dies erkennen könnte.

Da es jedoch nicht gelang, das Elixir-Programm in Verbindung mit der libiotrace zu starten, musste dieser Ansatz aufgegeben werden. Stattdessen wurden für das Schreiben erster Flux-Queries zum Erstellen von IO-Statistiken die vorhandenen Test-Scripts genutzt, für realistischere Ergebnisse können dann umfangreichere Programme wie der OpenFOAM Motorbike-Test verwendet werden. Sollten zu einem späteren Zeitpunkt doch eigene Testprogramme benötigt werden, um beispielsweise bestimmte IO-Muster zu simulieren, können diese in C oder Python implementiert werden.

3.2 IO-Statistiken

Zur Auswertung des protokollierten File-IO werden Flux-Queries für das Abrufen folgender Statistiken erstellt:

- Insgesamt geschriebene Datenmenge pro Prozess
- Insgesamt geschriebene Datenmenge pro Prozess für einen bestimmten Funktionsaufruf (z.B. MPI_File_write)
- Insgesamt geschriebene Datenmenge aller Prozesse in eine Datei für einen bestimmten Funktionsaufruf (z.B. MPI_File_write)
- Anzahl der Dateizugriffe pro Prozess (jeweils schreibend und lesend)
- Zeit zwischen erstem und letztem Schreibzugriff eines Prozesses (auf beliebige Dateien)
- Durchschnittliche Anzahl der IO-Zugriffe pro Sekunde (jeweils pro Prozess)
- Anzahl der Zugriffe eines Prozesses auf eine Datei
- Summe der IO-Zeit jedes Prozesses (wahlweise nur lesend, nur schreibend oder beides)

Die Queries können mit leichten Anpassungen zur Anzeige der Statistiken im Grafana-Dashboard verwendet werden, zur Entwicklung und zum Debugging sind jedoch oftmals auch die in InfluxDB enthaltenen Visualisierungsoptionen hilfreich.

Abbildung 3.1 zeigt beispielsweise die Visualisierung der Abfrage *Durchschnittliche Anzahl der IO-Zugriffe pro Sekunde* in InfluxDB, die Daten wurden mit dem Test-Skript *MPI_file_io_random* erzeugt. Es ist zu erkennen, dass die meisten Prozesse weniger als 140 Zugriffe/Sekunde im Durchschnitt ausführen. Der Quellcode der Flux-Query ist in Listing A.2 zu finden.

Zur Gegenprüfung dieser Ergebnisse kann nun die IO-Zeit der einzelnen Prozesse visualisiert werden. Abbildung 3.2 zeigt das Ergebnis der Abfrage *Summe der IO-Zeit jedes Prozesses* in der InfluxDB-Visualisierung (Flux-Quellcode siehe Listing A.3). Diese zeigt, dass der Großteil der Prozesse eine eher lange IO-Zeit zwischen 2 und 3 Sekunden hat, während nur ein kleiner Teil der Prozesse eine IO-Zeit von einer Sekunde oder weniger hat. Dieses Ergebnis, in Verbindung mit der vorherigen Abfrage könnte darauf hindeuten, dass die Prozesse sich beim Schreiben in eine Datei teilweise gegenseitig blockieren und deshalb bei den meisten Prozessen die Anzahl der IO-Zugriffe eher klein und die mit IO verbrachte Zeit eher groß ist.

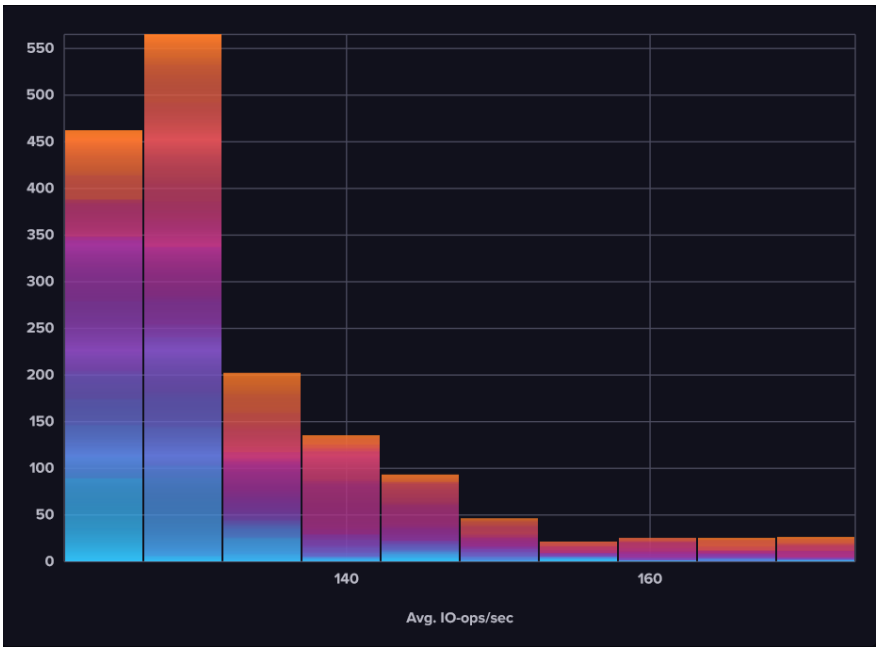

Abb. 3.1: Verteilung der IO-Aktivität einzelner Prozesse

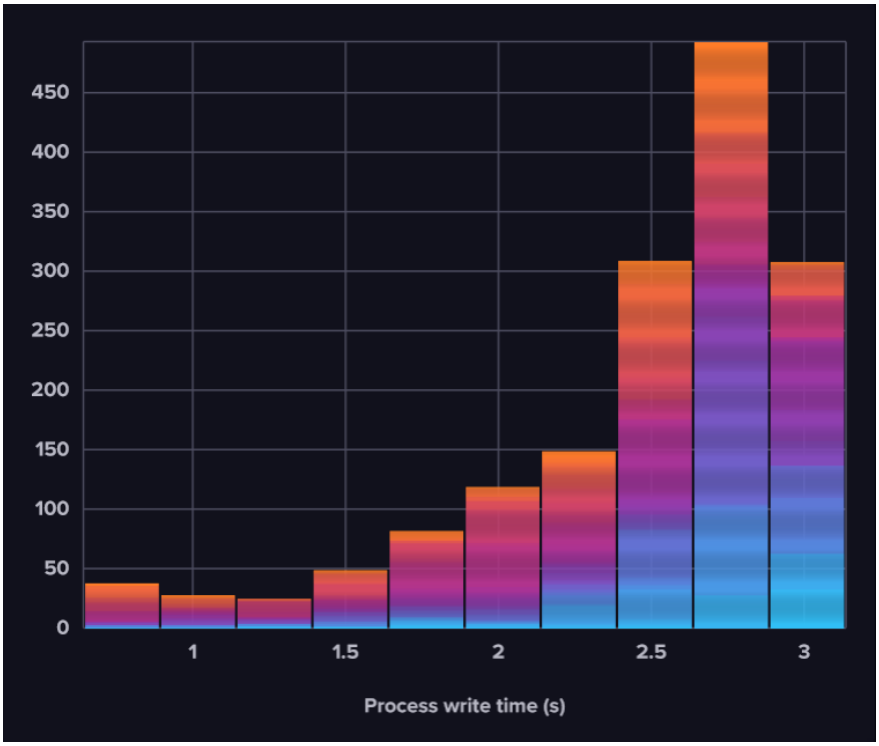

Abb. 3.2: Verteilung der IO-Zeit einzelner Prozesse

4 Ergebnis und Ausblick

Die Recherche zu bestehenden Lösungen zur IO-Charakterisierung in HPC-Anwendung ergab, dass die meisten Tools, die IO-Tracing ermöglichen, die daraus gewonnenen Daten entweder nur in geringem Umfang auswerten oder diese nur visuell aufbereiten, die Interpretation der Ergebnisse aber dann dem Nutzer überlassen bleibt. Lediglich Darshan erstellt aus den Tracing-Daten eine umfangreiche Statistik. Dies geschieht aber erst nach der vollständigen Programmausführung. Deshalb wurde beschlossen, mithilfe von Abfragen aus der Zeitreihendatenbank ähnliche Statistiken zur Laufzeit des Programms zu erstellen. Diese können kontinuierlich aktualisiert werden und ermöglichen es somit grundsätzlich, durch ineffizienten File-IO bedingte Performance-Probleme bei langlaufenden Programmen frühzeitiger zu erkennen.

Zur Erkennung von schlechtem IO-Mustern wurden mögliche Indikatoren, wie der gleichzeitige bzw. überlappende Zugriffsversuch mehrerer Prozesse auf eine einzelne Datei identifiziert. Mithilfe der aus der InfluxDB generierten Statistiken konnte in einem Test, bei dem viele Prozesse in dieselbe Datei schreiben wollen, ein möglicher IO-Bottleneck identifiziert werden.

Alternative Ansätze zur IO-Charakterisierung, wie die KI-basierte oder auf stochastischen Modellen beruhende Analyse von IO-Daten, die in einigen Papers erwähnt wurde, wurden vorerst verworfen, da ihre Funktionsweise oft schlecht nachvollziehbar ist und diese Methoden vermutlich auch gar nicht benötigt werden.

Im weiteren Verlauf des Forschungsprojekts sollten noch weitere Indikatoren für schlechten IO identifiziert werden, hierzu ist noch mehr Recherche zum Thema IO-Charakterisierung nötig, wenn möglich auch unabhängig von spezifischen Analyse-Tools, da diese oft sehr spezifisch auf einen Anwendungszweck zugeschnitten sind und wenig allgemeine Informationen liefern.

In Verbindung damit müssen eventuell noch weitere IO-Statistiken aus den libiotrace-Daten generiert werden. Ziel ist es hier, auch mit Hinblick auf die Anzeige der Daten im Grafana-Dashboard, ungefähr den Umfang und Detailgrad der Statistiken, die von Darshan und TAU erzeugt werden, zu erreichen. Außerdem soll bei der Entdeckung schlechter IO-Muster eine Entsprechende Meldung im Grafana-Dashboard angezeigt werden. Erste Ideen gehen hier in Richtung eines Logs, in dem Meldungen in Textform und entsprechend des Schweregrades in unterschiedlichen Farben angezeigt werden können.

Ein weiterer vielversprechender Ansatz besteht in einem Dateisystem-basierten Vorgehen: Statt die einzelnen Prozesse eines Programms als Ausgangspunkt der IO-Auswertung zu nutzen, wird betrachtet, mit welchen Dateien das Programm in welcher Form interagiert und auf welchem Dateisystem die Zugriffe erfolgen. So könnten ineffiziente Muster wie der häufige oder kleinteilige Zugriff auf vergleichsweise langsame Speichermeiden völlig unabhängig vom ausgeführten Programm erkannt werden.

Schlussendlich sollten, um die korrekte Erkennung von IO-Mustern zu validieren, zudem noch zusätzliche Testdaten bzw. Testprogramme erstellt werden, welche das entsprechende IO-Verhalten realistisch simulieren.

A Code-Listings

List. A.1: Elixir-Code zur Erzeugung von Testdaten für libiotrace

```
defmodule ElixirIO do
  @temp_dir "/tmp"
  @text_file @temp_dir <> "/text.txt"

  def start do
    for (i = 0; i < 10000;) {
      write_line(@text_file, "Line␣" <> i)
    }
  end

  defp write_line(path, content) do
    {_, workdir} = File.cwd()
    File.write(workdir <> path, content <> "\n", [:append])
  end

  def prepare() do
    {_, workdir} = File.cwd()
    File.mkdir(workdir <> @temp_dir)
  end

  def cleanup() do
    {_, workdir} = File.cwd()
    File.rm_rf(workdir <> @temp_dir)
  end
end

ElixirIO.prepare()
ElixirIO.start()
ElixirIO.cleanup()
```

List. A.2: Flux-Query zur Anzeige der Verteilung der IO-Aktivität von Prozessen

```
import "join"

data = from(bucket: "hsebucket")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "libiotrace")
  |> filter(fn: (r) => r["_field"] == "
    ↪ function_data_written_bytes" )

writes = data
  |> count()

first_write = data
  |> min(column: "_time")
  |> set(key: "mark", value: "first")

last_write = data
  |> max(column: "_time")
  |> set(key: "mark", value: "last")

duration = union(tables: [first_write, last_write])
  |> map(fn: (r) => ({r with timestamp: uint(v: r._time)}))
  |> pivot(rowKey: ["processid"], columnKey: ["mark"],
    ↪ valueColumn: "timestamp")
  |> map(fn: (r) => ({r with duration: r.last - r.first}))

join.inner(
  left: writes,
  right: duration,
  on: (l, r) => l.thread == r.thread,
  as: (l, r) => ({l with _value: l._value, duration: r.duration
    ↪ }),
)
  |> set(key:)
  |> map(fn: (r) => ({r with iosec: (float(v: r._value) / float
    ↪ (v: r.duration)) * float(v: 1000000000)}))
  |> yield(name: "Avg. IOPS")
```

List. A.3: Flux-Query zur Anzeige der Verteilung der IO-Zeit (schreibend) von Prozessen

```
from(bucket: "hsebucket")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "libiotrace")
  |> filter(fn: (r) => r.functionname =~ /.*write.*/)
  |> filter(fn: (r) => r._field == "time_diff")
  |> group(columns: ["hostname", "jobname", "processid"])
  |> sum(column: "_value")
  |> yield(name: "process_write_time")
```

Literatur

[1] Association for Computing Machinery. *ACM Digital Library*. besucht am 23.02.2024. 2024. URL: https://dl.acm.org/.

[2] Bitmotec GmbH. *InfluxDB – Einführung in die Open-Source-Zeitreihendatenbank*. besucht am 23.02.2024. 2023. URL: https://www.bitmotec.com/blog/influxdb-einfuehrung-in-die-open-source-zeitreihendatenbank/.

[3] Chao Chen u. a. „Decoupled I/O for Data-Intensive High Performance Computing“. In: *2014 43rd International Conference on Parallel Processing Workshops*. 2014, S. 312–320. DOI: 10.1109/ICPPW.2014.48.

[4] Google LLC. *Google Scholar*. besucht am 23.02.2024. 2024. URL: https://scholar.google.de/.

[5] InfluxData, Inc. *InfluxDB OSS v2 Documentation*. besucht am 23.02.2024. 2024. URL: https://docs.influxdata.com/influxdb/v2/.

[6] NASA Advanced Supercomputing Division. *Using Darshan for I/O Characterization*. besucht am 23.02.2024. 2022. URL: https://www.nas.nasa.gov/hecc/support/kb/using-darshan-for-io-characterization_681.html.

[7] Arifa Nisar, Wei-keng Liao und Alok Choudhary. „Delegation-Based I/O Mechanism for High Performance Computing Systems“. In: *IEEE Transactions on Parallel and Distributed Systems* 23.2 (2012), S. 271–279. DOI: 10.1109/TPDS.2011.166.

[8] Sameer Shende u. a. „Characterizing I/O performance using the TAU performance system“. In: *Applications, Tools and Techniques on the Road to Exascale Computing*. IOS Press, 2012, S. 647–655.

[9] Shane Snyder u. a. „Modular HPC I/O characterization with Darshan“. In: *Proceedings of the 5th Workshop on Extreme-Scale Programming Tools*. ESPT '16. Salt Lake City, Utah: IEEE Press, 2016, 9–17. ISBN: 9781509039180.

[10] Karthik Vijayakumar u. a. „Scalable I/O tracing and analysis“. In: *Proceedings of the 4th Annual Workshop on Petascale Data Storage*. PDSW '09. Portland, Oregon: Association for Computing Machinery, 2009, 26–31. ISBN: 9781605588834. DOI: 10.1145/1713072.1713080. URL: https://doi.org/10.1145/1713072.1713080.