technische universität
dortmund

Leistungsanalyse der Datenstromverarbeitung in kCQL

Tim Tannert

Lehrstuhl für Informatik 12
TU Dortmund

28. Februar 2017

1 Einführung in kCQL

2 Vorbereitung der Leistungsanalyse

3 Analyse der Systemlast

4 Analyse der Verarbeitungsgeschwindigkeit

5 Zusammenfassung

6 Quellen

technische universität
dortmund

ESS

1 Einführung in kCQL

- Allgemeines
- Funktionsweise
- Motiviation der Leistungsanalyse

technische universität
dortmund

ESS

Entwicklung

- Stellt eine **deklarative Schnittstelle** zum Linuxkernel bereit.

technische universität
dortmund

ESS

Entwicklung

- Stellt eine **deklarative Schnittstelle** zum Linuxkernel bereit.
- Zugriff auf **Zustandsdaten und Ereignisse** aus dem Linuxkernel.

technische universität dortmund

Entwicklung

- Stellt eine **deklarative Schnittstelle** zum Linuxkernel bereit.
- Zugriff auf **Zustandsdaten und Ereignisse** aus dem Linuxkernel.
  - Einheitlicher und leichterer Zugriff (z.B. für Programmierer, Systemadministratoren, ...).

technische universität dortmund

ESS

Entwicklung

- Stellt eine **deklarative Schnittstelle** zum Linuxkernel bereit.
- Zugriff auf **Zustandsdaten und Ereignisse** aus dem Linuxkernel.
  - Einheitlicher und leichterer Zugriff (z.B. für Programmierer, Systemadministratoren, ...).
- Eigenentwicklung der Arbeitsgruppe ESS

technische universität dortmund

ESS

Entwicklung

- Stellt eine **deklarative Schnittstelle** zum Linuxkernel bereit.
- Zugriff auf **Zustandsdaten und Ereignisse** aus dem Linuxkernel.
  - Einheitlicher und leichterer Zugriff (z.B. für Programmierer, Systemadministratoren, ...).
- Eigenentwicklung der Arbeitsgruppe ESS
  - Die verwendete Anfragesprache *Continuous Query Language* (CQL) wurde in Stanford entwickelt.

technische universität dortmund

ESS

Entwicklung

- Stellt eine **deklarative Schnittstelle** zum Linuxkernel bereit.
- Zugriff auf **Zustandsdaten und Ereignisse** aus dem Linuxkernel.
  - Einheitlicher und leichterer Zugriff (z.B. für Programmierer, Systemadministratoren, ...).
- Eigenentwicklung der Arbeitsgruppe ESS
  - Die verwendete Anfragesprache *Continuous Query Language* (CQL) wurde in Stanford entwickelt.
  - Die Verwendung von CQL (C++) als Kernelmodul wird durch Mark Veltzer's Framework *kcpp* ermöglicht .

Anfragen

- kCQL-Anfragen basieren auf CQL-Anfragen.

Anfragen

- kCQL-Anfragen basieren auf CQL-Anfragen.
  - Sie ähneln SQL-Anfragen.

Anfragen

- kCQL-Anfragen basieren auf CQL-Anfragen.
  - Sie ähneln SQL-Anfragen.
  - Zusätzlich zu Relationen werden Datenströme unterstützt.

Anfragen

- kCQL-Anfragen basieren auf CQL-Anfragen.
  - Sie ähneln SQL-Anfragen.
  - Zusätzlich zu Relationen werden Datenströme unterstützt.
- CQL kann nicht direkt mit Datenströmen umgehen (z.B. zwei Datenströme joinen).

Anfragen

- kCQL-Anfragen basieren auf CQL-Anfragen.
  - Sie ähneln SQL-Anfragen.
  - Zusätzlich zu Relationen werden Datenströme unterstützt.
- CQL kann nicht direkt mit Datenströmen umgehen (z.B. zwei Datenströme joinen).
  - Datenströmen müssen in Relationen überführt werden.

technische universität dortmund

Anfragen

- kCQL-Anfragen basieren auf CQL-Anfragen.
  - Sie ähneln SQL-Anfragen.
  - Zusätzlich zu Relationen werden Datenströme unterstützt.
- CQL kann nicht direkt mit Datenströmen umgehen (z.B. zwei Datenströme joinen).
  - Datenströmen müssen in Relationen überführt werden.

Netsimple-Anfrage [SLS15]

```
Netsimple : RSTREAM (
   SELECT process.name , net.macprot , net.datalength
   FROM net[now] , sockets , process
   WHERE net.socket = sockets.sid
     AND sockets.pid = process.pid ;
);
```

Anfrageplan

technische universität dortmund

Anfrageplan

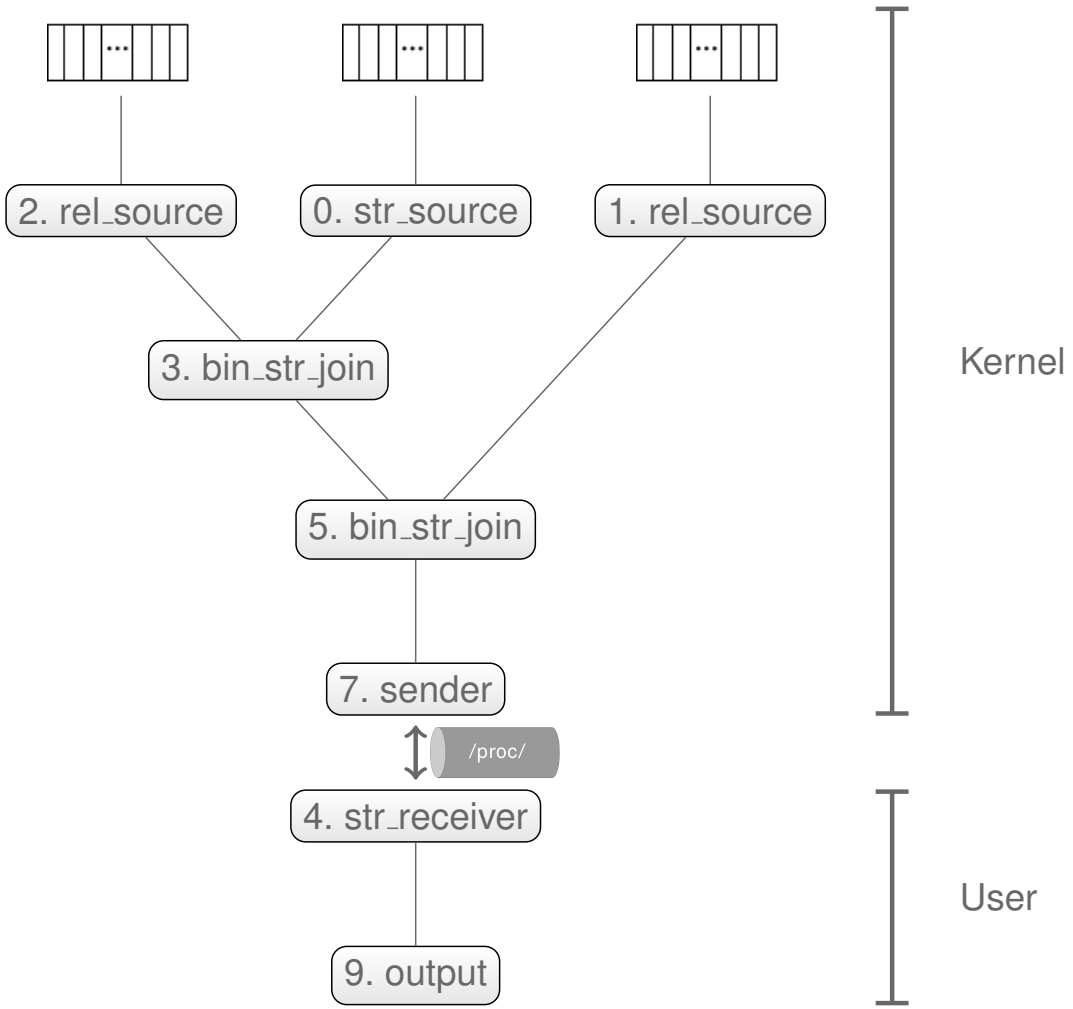


Abbildung: Abstrakte Darstellung des Anfrageplans, der *Netsimple*-Anfrage.

technische universität
dortmund

ESS

Offene Probleme

technische universität dortmund

Offene Probleme

- Die Untersuchung bezieht sich lediglich auf die Systemlast.
  - Keine Aussage über die Verarbeitungsgeschwindigkeit!

Offene Probleme

- Die Untersuchung bezieht sich lediglich auf die Systemlast.
  - Keine Aussage über die Verarbeitungsgeschwindigkeit!
- Die Geschwindigkeit von kCQL könnte größer sein, als die der alternativen Werkzeuge.

Offene Probleme

- Die Untersuchung bezieht sich lediglich auf die Systemlast.
  - Keine Aussage über die Verarbeitungsgeschwindigkeit!
- Die Geschwindigkeit von kCQL könnte größer sein, als die der alternativen Werkzeuge.

$\Rightarrow$ Eine **Lastanalyse** zur Identifikation lastintensiver Programmteile ist wünschenswert.
$\Rightarrow$ Eine genauere **Verarbeitungsgeschwindigkeitsanalyse** ist empfehlenswert.

2 Vorbereitung der Leistungsanalyse

- Paradigmen der Leistungsanalyse
- Überblick über verfügbare Werkzeuge

Ansätze der Softwareuntersuchung

1. **Untersuchung zur Laufzeit** (*Profiling*):

Ansätze der Softwareuntersuchung

1. **Untersuchung zur Laufzeit** (*Profiling*):
   - Aufzeichnung bestimmter Ereignisse mithilfe von *Profilingwerkzeugen*.

Ansätze der Softwareuntersuchung

1. **Untersuchung zur Laufzeit** (*Profiling*):
   - Aufzeichnung bestimmter Ereignisse mithilfe von *Profilingwerkzeugen*.
     - Dadurch ist beispielsweise eine Funktions- und Speicherbetrachtung möglich.

Ansätze der Softwareuntersuchung

1. **Untersuchung zur Laufzeit** (*Profiling*):
   - Aufzeichnung bestimmter Ereignisse mithilfe von *Profilingwerkzeugen*.
     - Dadurch ist beispielsweise eine Funktions- und Speicherbetrachtung möglich.
   - Der Quellcode des zu untersuchenden Programmes ist nicht notwendig.

Ansätze der Softwareuntersuchung

1. **Untersuchung zur Laufzeit** (*Profiling*):
   - Aufzeichnung bestimmter Ereignisse mithilfe von *Profilingwerkzeugen*.
     - Dadurch ist beispielsweise eine Funktions- und Speicherbetrachtung möglich.
   - Der Quellcode des zu untersuchenden Programmes ist nicht notwendig.
2. **Manuelle Instrumentierung** (*Tracing*):

Ansätze der Softwareuntersuchung

1. **Untersuchung zur Laufzeit** (*Profiling*):
   - Aufzeichnung bestimmter Ereignisse mithilfe von *Profilingwerkzeugen*.
     - Dadurch ist beispielsweise eine Funktions- und Speicherbetrachtung möglich.
   - Der Quellcode des zu untersuchenden Programmes ist nicht notwendig.
2. **Manuelle Instrumentierung** (*Tracing*):
   - Einbauen von *Tracepoints* in das zu untersuchende Programm.

Ansätze der Softwareuntersuchung

1. **Untersuchung zur Laufzeit** (*Profiling*):
   - Aufzeichnung bestimmter Ereignisse mithilfe von *Profilingwerkzeugen*.
     - Dadurch ist beispielsweise eine Funktions- und Speicherbetrachtung möglich.
   - Der Quellcode des zu untersuchenden Programmes ist nicht notwendig.
2. **Manuelle Instrumentierung** (*Tracing*):
   - Einbauen von *Tracepoints* in das zu untersuchende Programm.
   - Anschließendes Ausführen des Programmes.

Ansätze der Softwareuntersuchung

1. **Untersuchung zur Laufzeit** (*Profiling*):
   - Aufzeichnung bestimmter Ereignisse mithilfe von *Profilingwerkzeugen*.
     - Dadurch ist beispielsweise eine Funktions- und Speicherbetrachtung möglich.
   - Der Quellcode des zu untersuchenden Programmes ist nicht notwendig.
2. **Manuelle Instrumentierung** (*Tracing*):
   - Einbauen von *Tracepoints* in das zu untersuchende Programm.
   - Anschließendes Ausführen des Programmes.
   - Das Vorhandensein des Quellcodes ist essentiell wichtig.

technische universität
dortmund

ESS

Verglichene Werkzeuge

Verglichene Werkzeuge

Untersuchung zur Laufzeit	Manuelle Instrumentierung
Perf	Perf
oProfile	LTTng
SystemTap	-
SysProf	-
Offcputime	-
PAPI	-

Tabelle: Zuordnung der Werkzeuge zu den Analysestrategien

technische universität
dortmund

ESS

3 Analyse der Systemlast

- Allgemeines Vorgehen
- Ergebnisse

Weitere Vorüberlegung

- Eine Metrik für die Messung der erzeugten Systemlast muss gefunden werden.

Weitere Vorüberlegung

- Eine Metrik für die Messung der erzeugten Systemlast muss gefunden werden.
  - Die Häufigkeit und Dauer von Funktionsaufrufen sind eine messbare Metrik für die erzeugte Systemlast

Weitere Vorüberlegung

- Eine Metrik für die Messung der erzeugten Systemlast muss gefunden werden.
  - Die Häufigkeit und Dauer von Funktionsaufrufen sind eine messbare Metrik für die erzeugte Systemlast

$\Rightarrow$ **Perf** und **oProfile** sind für eine Analyse der Systemlast geeignet.

Weitere Vorüberlegung

- Eine Metrik für die Messung der erzeugten Systemlast muss gefunden werden.
  - Die Häufigkeit und Dauer von Funktionsaufrufen sind eine messbare Metrik für die erzeugte Systemlast

$\Rightarrow$ **Perf** und **oProfile** sind für eine Analyse der Systemlast geeignet.

Auswahl fällt auf **Perf**, da dessen Ausgaben mit *Hot-Graphs* visualisiert werden können!

technische universität
dortmund

Vorgehen

1. Auswahl der zu untersuchenden Anfrage:

technische universität dortmund

ESS

Vorgehen

1. Auswahl der zu untersuchenden Anfrage:

Netsimple-Anfrage

SELECT process.name,net.macprot,net.datalength
FROM net [now],sockets,process
WHERE net.socket=sockets.sid and sockets.pid=process.pid;

Vorgehen

2. Bestimmung passender interner Last:

Vorgehen

2. Bestimmung passender interner Last:

- Echte Last:
  - 10Mbit
  - 100Mbit
  - 800Mbit

Vorgehen

2. Bestimmung passender interner Last:

- Echte Last:
  - 10Mbit
  - 100Mbit
  - 800Mbit
- Künstliche Last:
  - 833Hz
  - 8333Hz
  - 66667Hz

technische universität
dortmund

Vorgehen

3. Erzeugung der *Hot-Graphs*:

Vorgehen

3 Erzeugung der *Hot-Graphs*:

- Mit/Ohne externer Last (Linux-Kernel-Übersetzung im Hintergrund)

Vorgehen

3 Erzeugung der *Hot-Graphs*:

- Mit/Ohne externer Last (Linux-Kernel-Übersetzung im Hintergrund)
- Echte/Unechte interne Last. (Last für kCQL)

Vorgehen

3 Erzeugung der *Hot-Graphs*:

- Mit/Ohne externer Last (Linux-Kernel-Übersetzung im Hintergrund)
- Echte/Unechte interne Last. (Last für kCQL)
- In der Arbeit aus Zeitgründen nur 5 Stück für jedes Lastszenario, mit anschließender Auswahl eines Medians.
  - **Hier 300 Stück für jedes Lastszenario**

technische universität dortmund

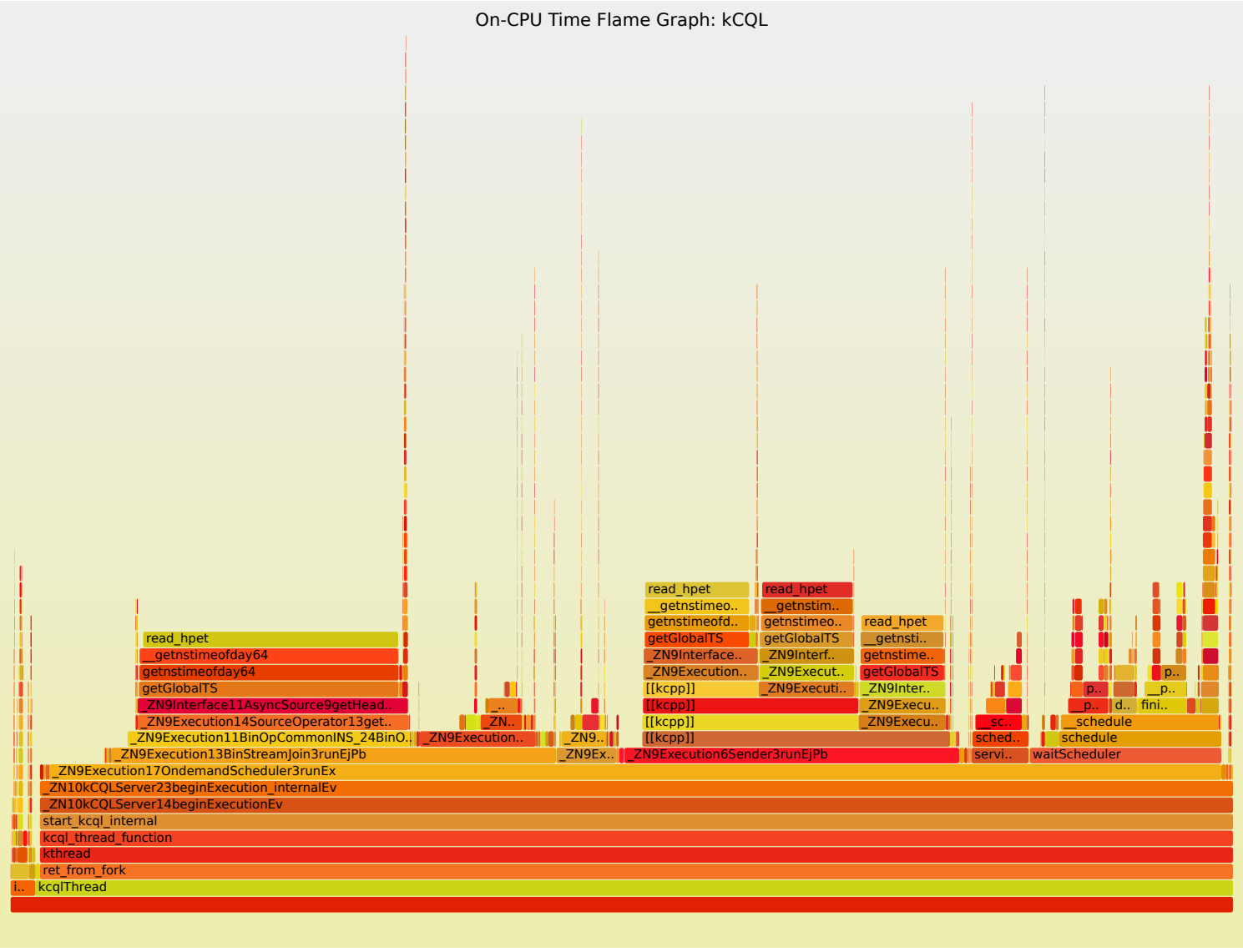
Vorgehen

4. Betrachtung der Funktionen, die am häufigsten ausgeführt werden.

technische universität dortmund

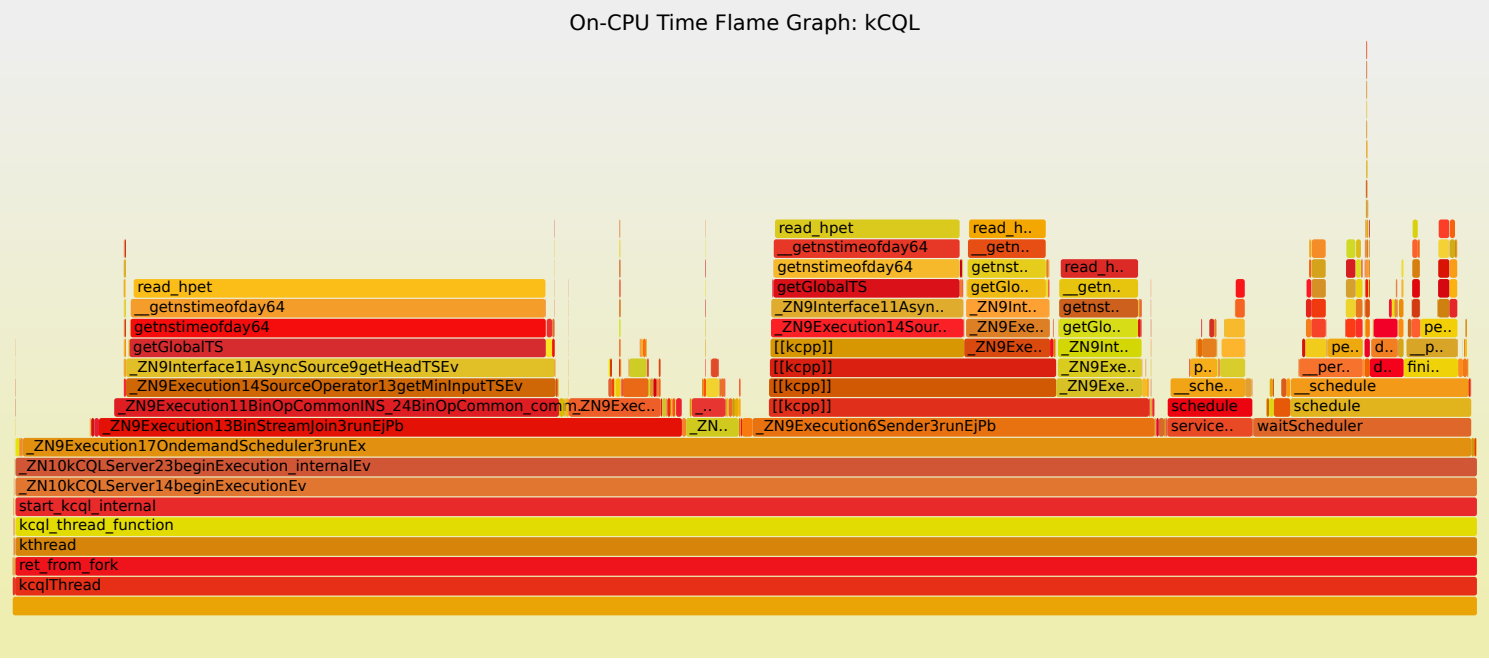
ESS

Hot-Graph ohne externe Last (100Mbit)



technische universität dortmund

Hot-Graph ohne externe Last (8333Hz)



Optimierbare Programmteile

- Die vom Userteil erzeugte Last, ist um ein vielfaches geringer, als die des Kernel-Moduls. $\rightarrow$ **Ignorieren**

Optimierbare Programmteile

- Die vom Userteil erzeugte Last, ist um ein vielfaches geringer, als die des Kernel-Moduls. $\rightarrow$ **Ignorieren**
- Auffällige Funktionen sind ***getGlobalTS***, ***service_schedule*** und ***waitScheduler***.

Optimierbare Programmteile

- Die vom Userteil erzeugte Last, ist um ein vielfaches geringer, als die des Kernel-Moduls. $\rightarrow$ **Ignorieren**
- Auffällige Funktionen sind ***getGlobalTS***, ***service_schedule*** und ***waitScheduler***.

Eine ausgewogene Mischung der Vermeidung und der Optimierung ist vermutlich das beste Vorgehen.

Weitere Ergebnisse:

- Bei echter interner Last, ist die Anzahl der absoluten Funktionsaufrufe größer.

technische universität dortmund

Weitere Ergebnisse:

- Bei echter interner Last, ist die Anzahl der absoluten Funktionsaufrufe größer.
- Mit Dummy-Daten ist die relative Last des Userteils geringer.

Weitere Ergebnisse:

- Bei echter interner Last, ist die Anzahl der absoluten Funktionsaufrufe größer.
- Mit Dummy-Daten ist die relative Last des Userteils geringer.
- Bei großer interner Last erzeugt ***service_schedule*** viel Last.

technische universität
dortmund

ESS

Weitere Ergebnisse:

- Bei echter interner Last, ist die Anzahl der absoluten Funktionsaufrufe größer.
- Mit Dummy-Daten ist die relative Last des Userteils geringer.
- Bei großer interner Last erzeugt ***service_schedule*** viel Last.
- ...

technische universität
dortmund

ESS

4 Analyse der Verarbeitungsgeschwindigkeit

- Allgemeines Vorgehen
- Ergebnisse

Weitere Vorüberlegung

- Die Zeit, die eine Anfrage benötigt, um von Quelle zu Senke zu gelangen, hängt vom erzeugten Anfrageplan ab.

Weitere Vorüberlegung

- Die Zeit, die eine Anfrage benötigt, um von Quelle zu Senke zu gelangen, hängt vom erzeugten Anfrageplan ab.
  - Eine Untersuchung der einzelnen Teile eines Anfrageplans ist ideal.

Weitere Vorüberlegung

- Die Zeit, die eine Anfrage benötigt, um von Quelle zu Senke zu gelangen, hängt vom erzeugten Anfrageplan ab.
  - Eine Untersuchung der einzelnen Teile eines Anfrageplans ist ideal.

$\Rightarrow$ Das Traceingwerkzeug **LTTng** ist für die genauere Untersuchung des Anfrageplans geeignet.

technische universität dortmund

Vorgehen

1. Auswahl der zu untersuchenden Anfrage:

technische universität dortmund

Vorgehen

1. Auswahl der zu untersuchenden Anfrage:

Auch in diesem Fall wird die **Netsimple-Anfrage** untersucht.

technische universität dortmund

Vorgehen

2. Instrumentierung aller vorhandener Operatoren:

technische universität dortmund

Vorgehen

2 Instrumentierung aller vorhandener Operatoren:

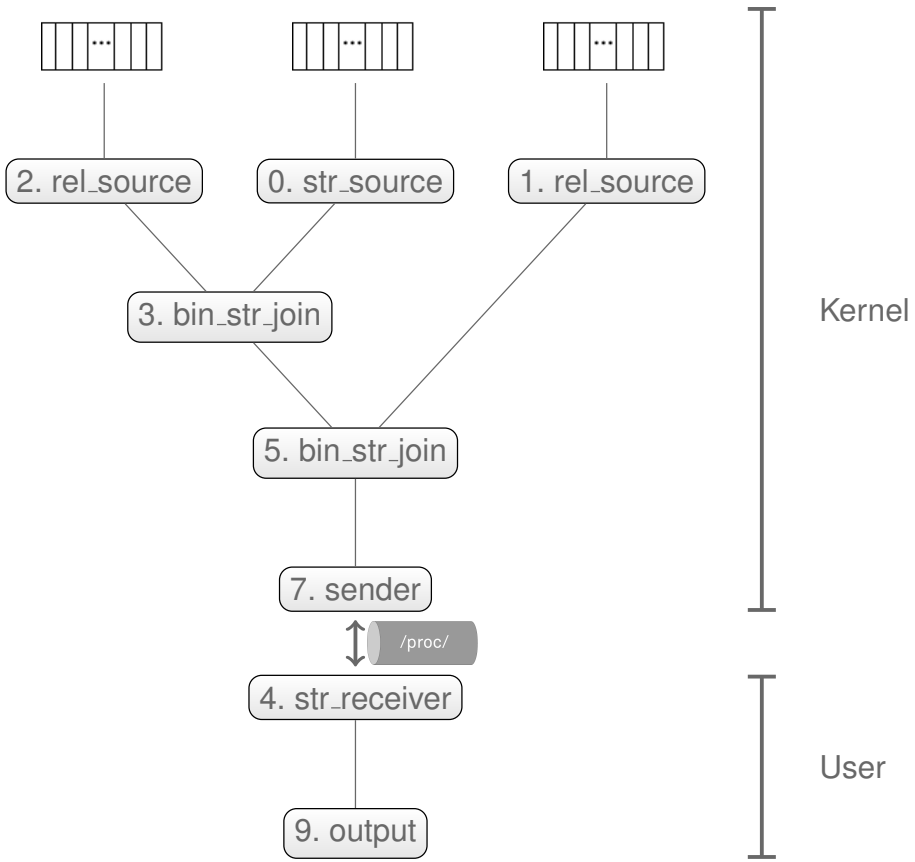


Abbildung: Anfrageplan der **Netsimple-Anfrage**.

technische universität dortmund

Vorgehen

3. Auswahl geeigneter Last:

technische universität dortmund

Vorgehen

3. Auswahl geeigneter Last:

- In dieser Versuchsreihe nur künstliche Last.

technische universität dortmund

Vorgehen

3. Auswahl geeigneter Last:

- In dieser Versuchsreihe nur künstliche Last.
  - Da diese besser zu steuern und nicht limitiert ist.

Vorgehen

3 Auswahl geeigneter Last:

- In dieser Versuchsreihe nur künstliche Last.
  - Da diese besser zu steuern und nicht limitiert ist.
- Insgesamt werden je Lastszenario **50000 Pakete** verarbeitet.

technische universität dortmund

Vorgehen

3 Auswahl geeigneter Last:

- In dieser Versuchsreihe nur künstliche Last.
  - Da diese besser zu steuern und nicht limitiert ist.
- Insgesamt werden je Lastszenario **50000 Pakete** verarbeitet.
- Die Durchlaufzeiten und Teildurchlaufzeiten bei **1000Hz/10000Hz/100000Hz** werden gemessen.

technische universität dortmund

Vorgehen

3 Auswahl geeigneter Last:

- In dieser Versuchsreihe nur künstliche Last.
  - Da diese besser zu steuern und nicht limitiert ist.
- Insgesamt werden je Lastszenario **50000 Pakete** verarbeitet.
- Die Durchlaufzeiten und Teildurchlaufzeiten bei **1000Hz/10000Hz/100000Hz** werden gemessen.
- Zusätzlich wird ein Szenario mit Laststößen untersucht.

technische universität dortmund

ESS

Vorgehen

3 Auswahl geeigneter Last:

- In dieser Versuchsreihe nur künstliche Last.
  - Da diese besser zu steuern und nicht limitiert ist.
- Insgesamt werden je Lastszenario **50000 Pakete** verarbeitet.
- Die Durchlaufzeiten und Teildurchlaufzeiten bei **1000Hz/10000Hz/100000Hz** werden gemessen.
- Zusätzlich wird ein Szenario mit Laststößen untersucht.
  - Jede Sekunde werden **1000 Pakete** als Laststoß in die kCQL-Eingabe geschrieben.

Erwartetes Verhalten

- Im Optimalfall schwankt die Verarbeitungsgeschwindigkeit nicht in Abhängigkeit der internen Last.

Erwartetes Verhalten

- Im Optimalfall schwankt die Verarbeitungsgeschwindigkeit nicht in Abhängigkeit der internen Last.
  - In der Realität ist dies aber nicht so!

technische universität
dortmund

ESS

Allgemeine Ergebnisse

- Die Verarbeitungszeiten der Operatoren sind deutlich geringer, als die Transportzeiten.

Allgemeine Ergebnisse

- Die Verarbeitungszeiten der Operatoren sind deutlich geringer, als die Transportzeiten.
  - **Diese sind somit zu vernachlässigen!**

Allgemeine Ergebnisse

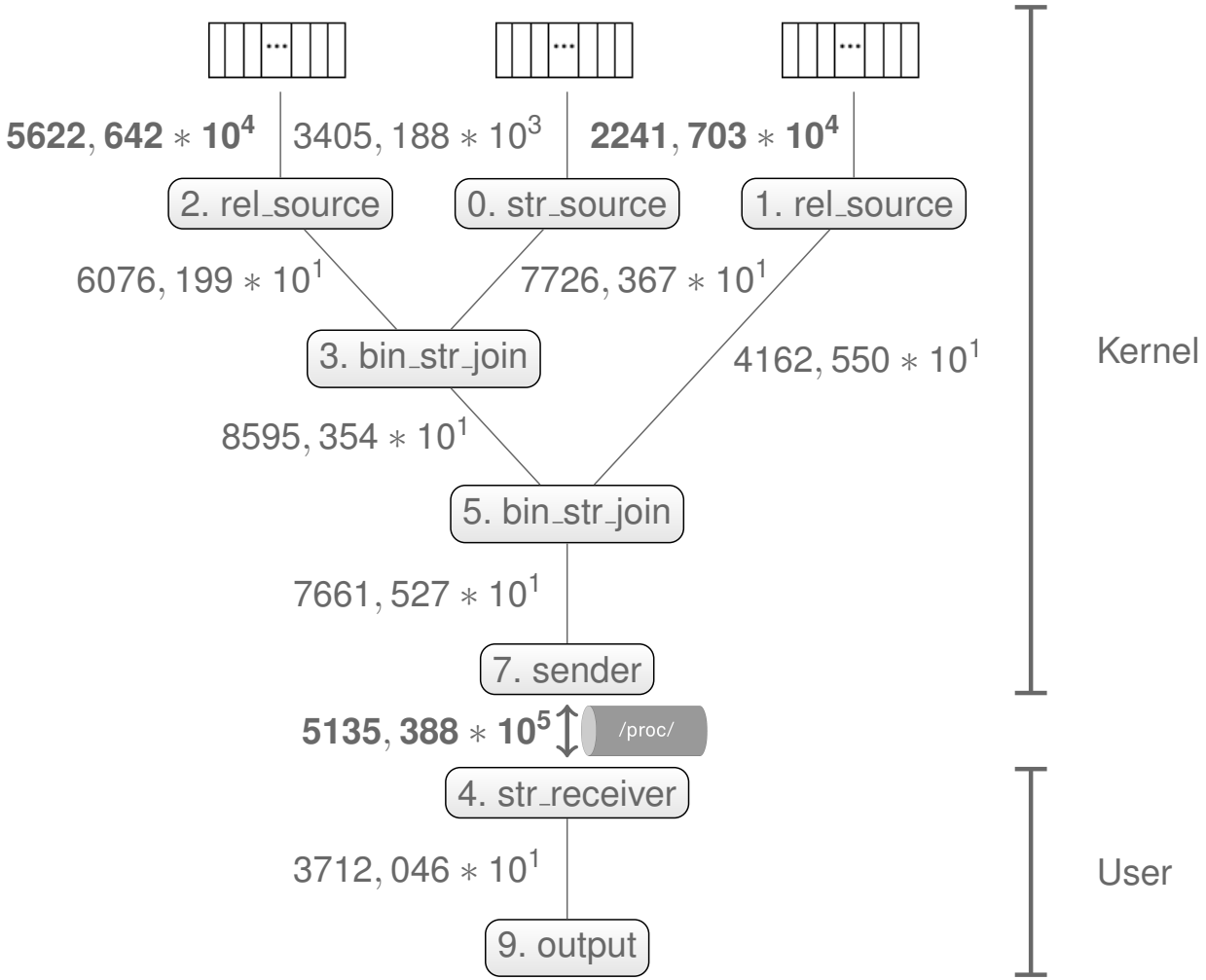
- Die Verarbeitungszeiten der Operatoren sind deutlich geringer, als die Transportzeiten.
  - **Diese sind somit zu vernachlässigen!**
- Der Fokus der Betrachtung liegt auf den Transportzeiten.

Allgemeine Ergebnisse

- Die Verarbeitungszeiten der Operatoren sind deutlich geringer, als die Transportzeiten.
  - **Diese sind somit zu vernachlässigen!**
- Der Fokus der Betrachtung liegt auf den Transportzeiten.

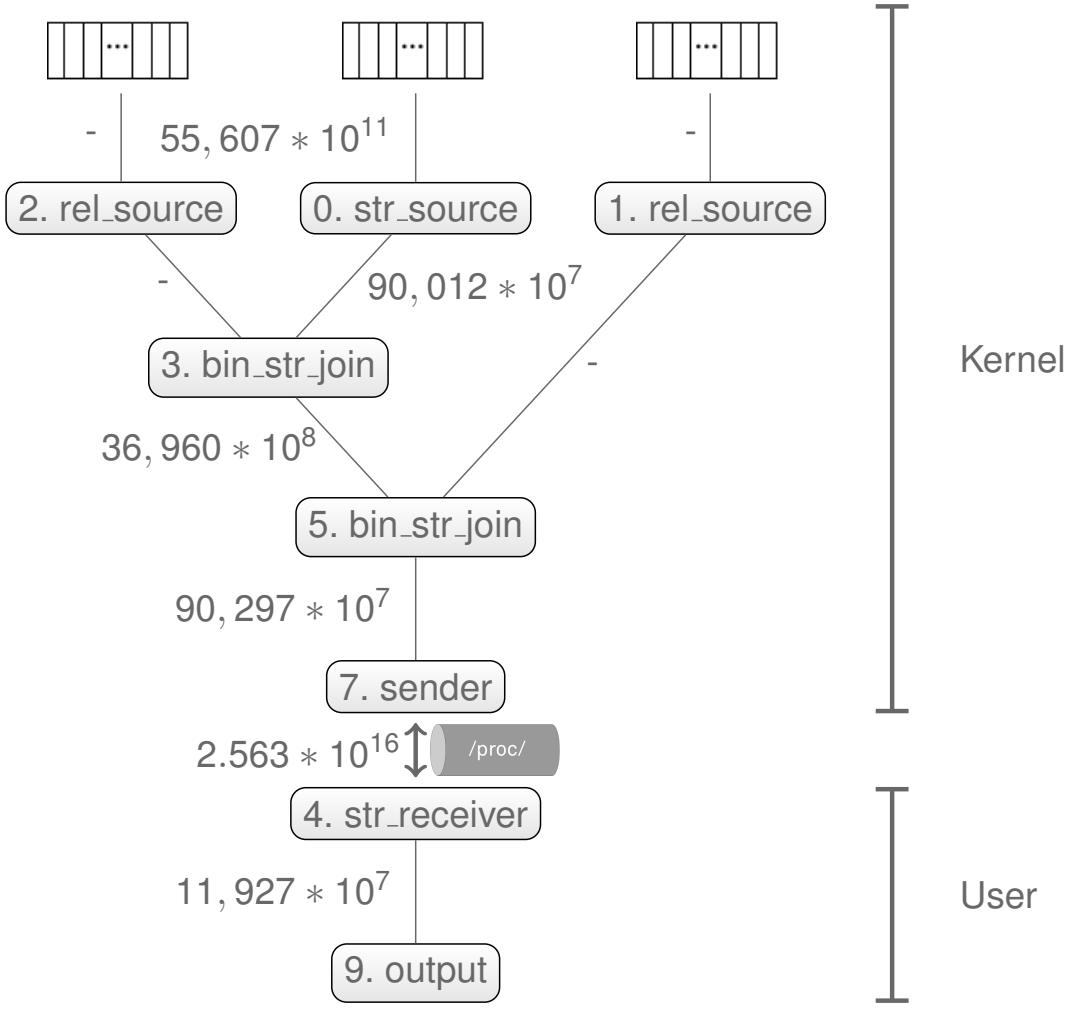
technische universität dortmund

Transportzeiten in Nanosekunden (Datenstöße)



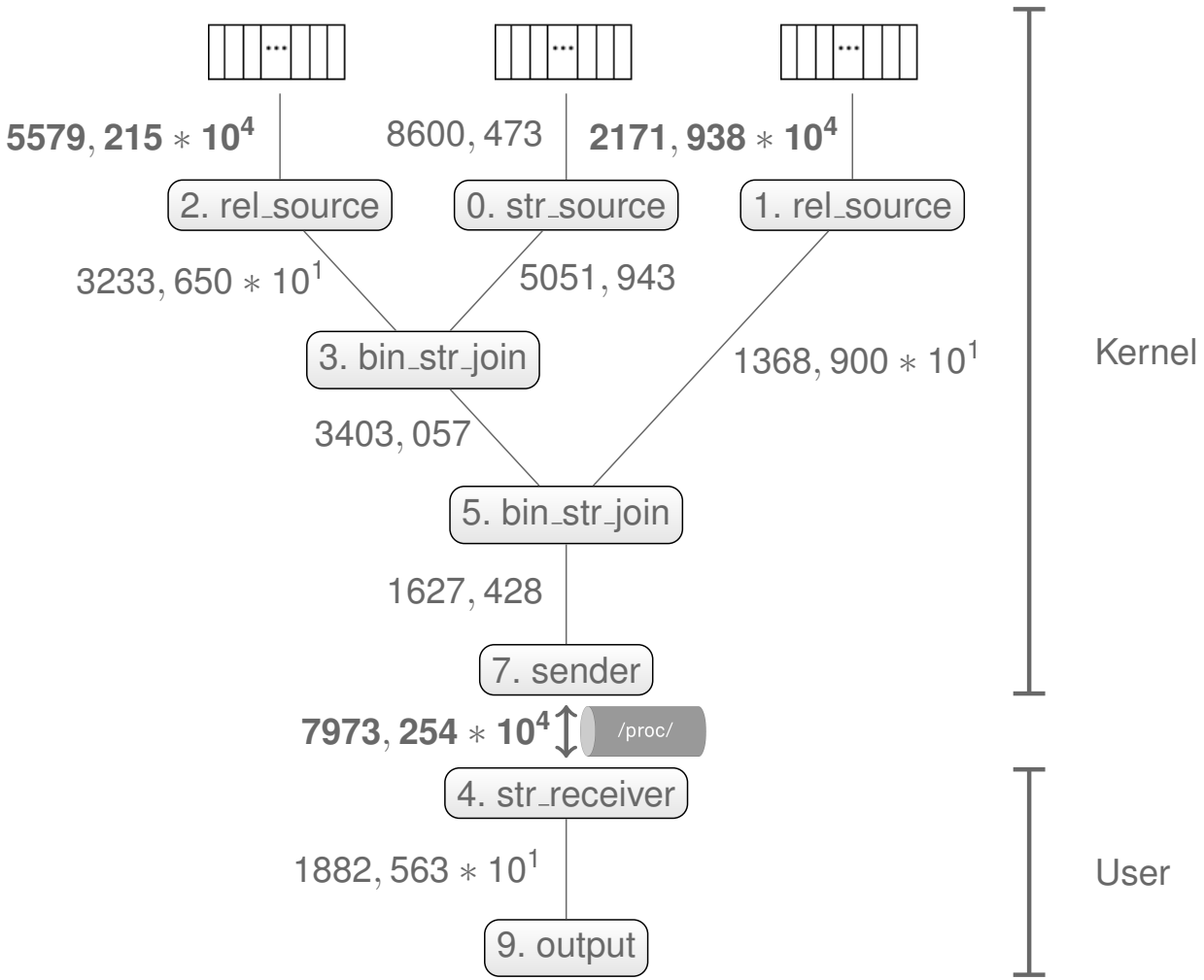
technische universität dortmund

Streuung (Datenstöße)



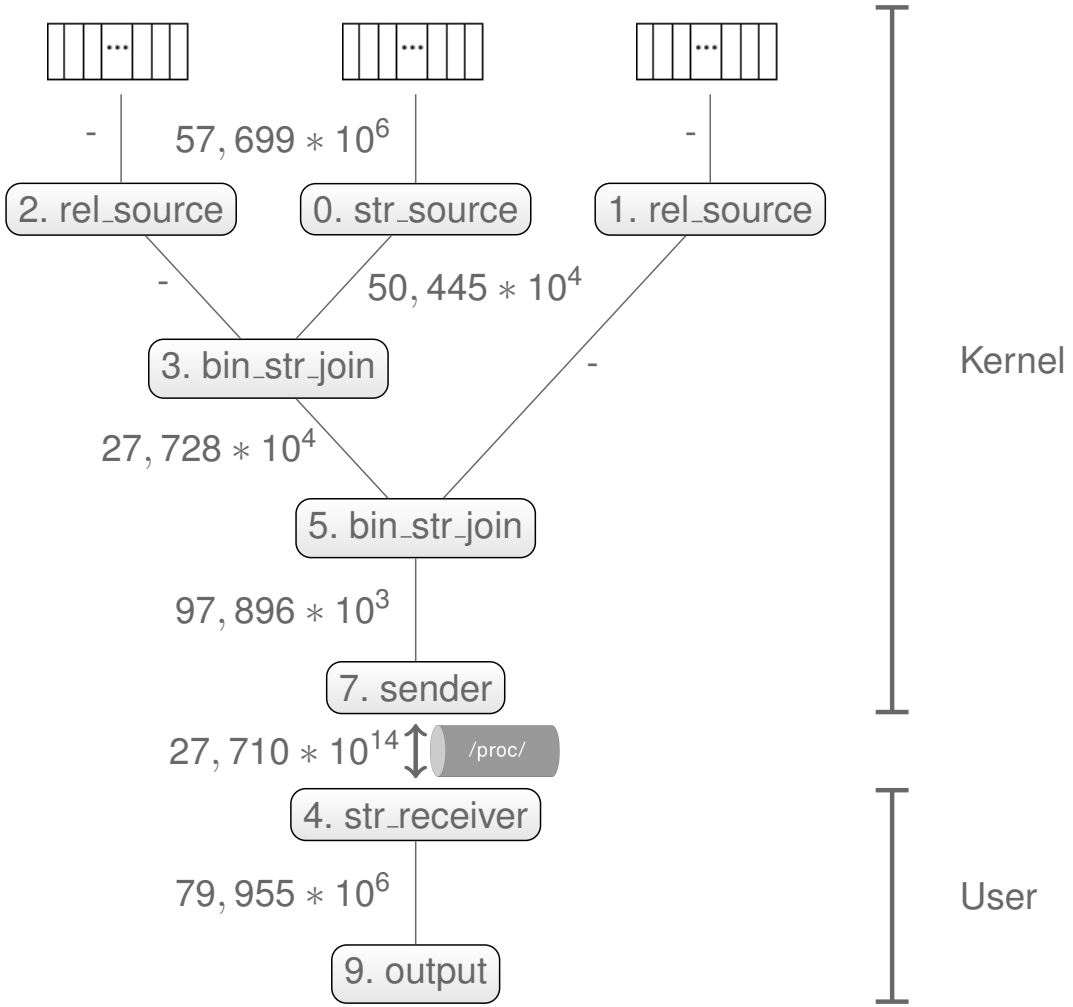
technische universität dortmund

Transportzeiten in Nanosekunden (1000Hz)



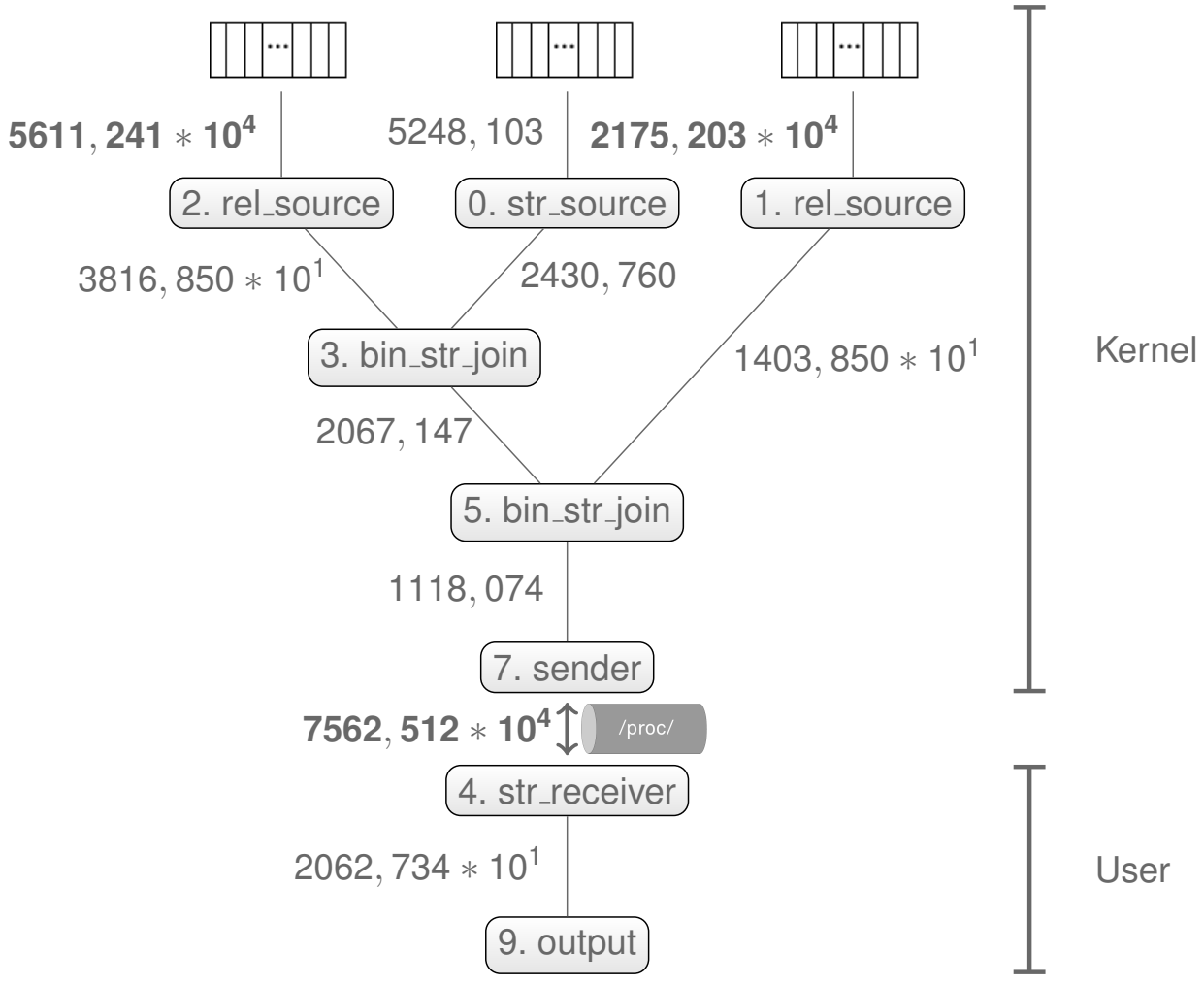
technische universität dortmund

Streuung (1000Hz)



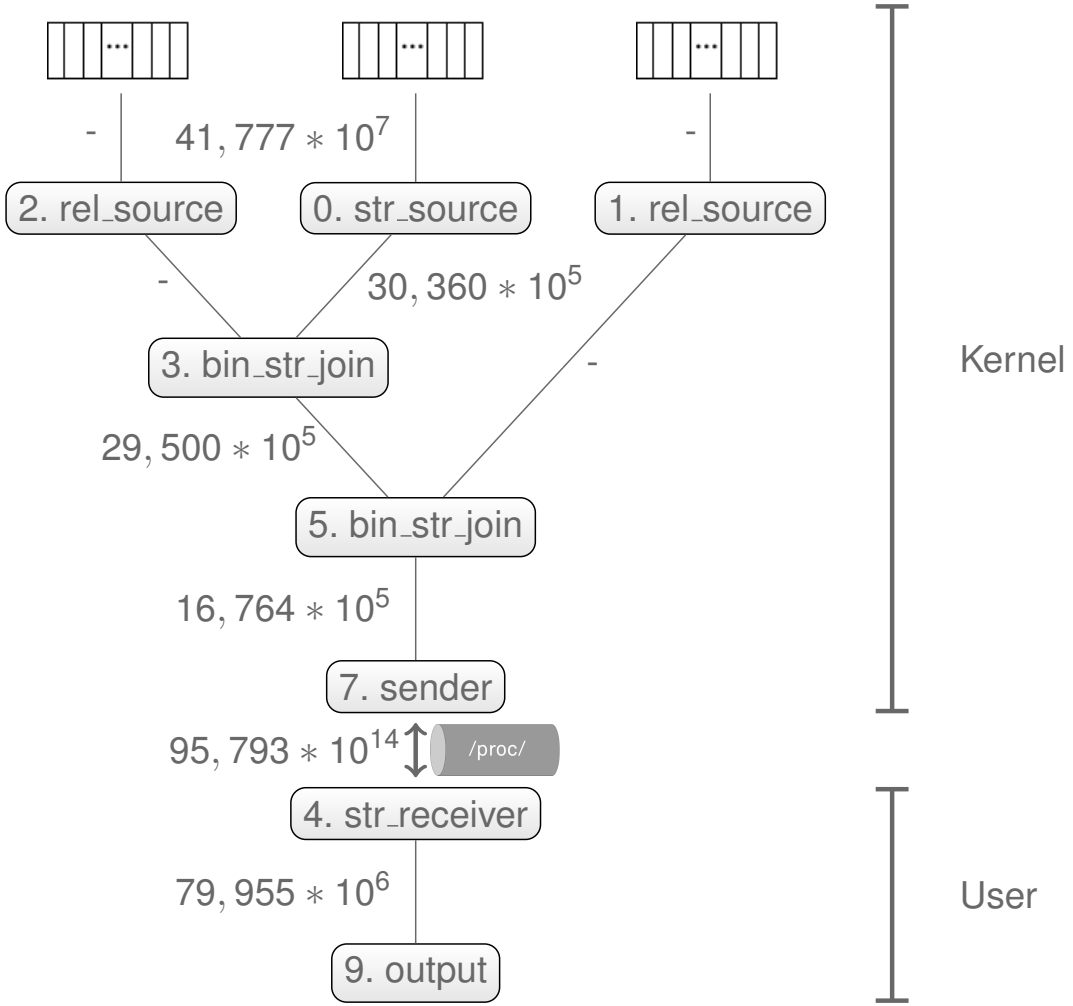
technische universität dortmund

Transportzeiten in Nanosekunden (10000Hz)



technische universität dortmund

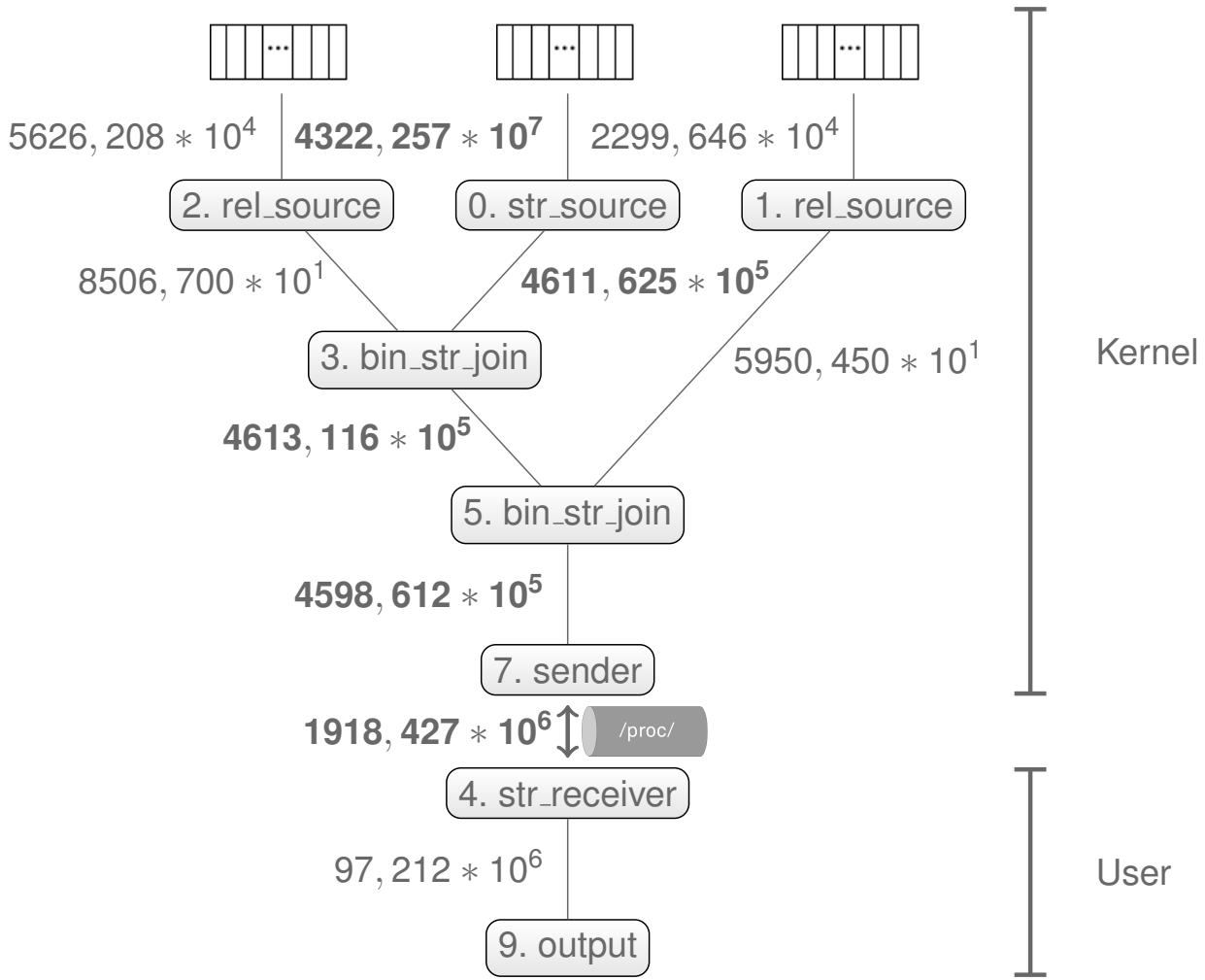
Streuung (10000Hz)



technische universität dortmund

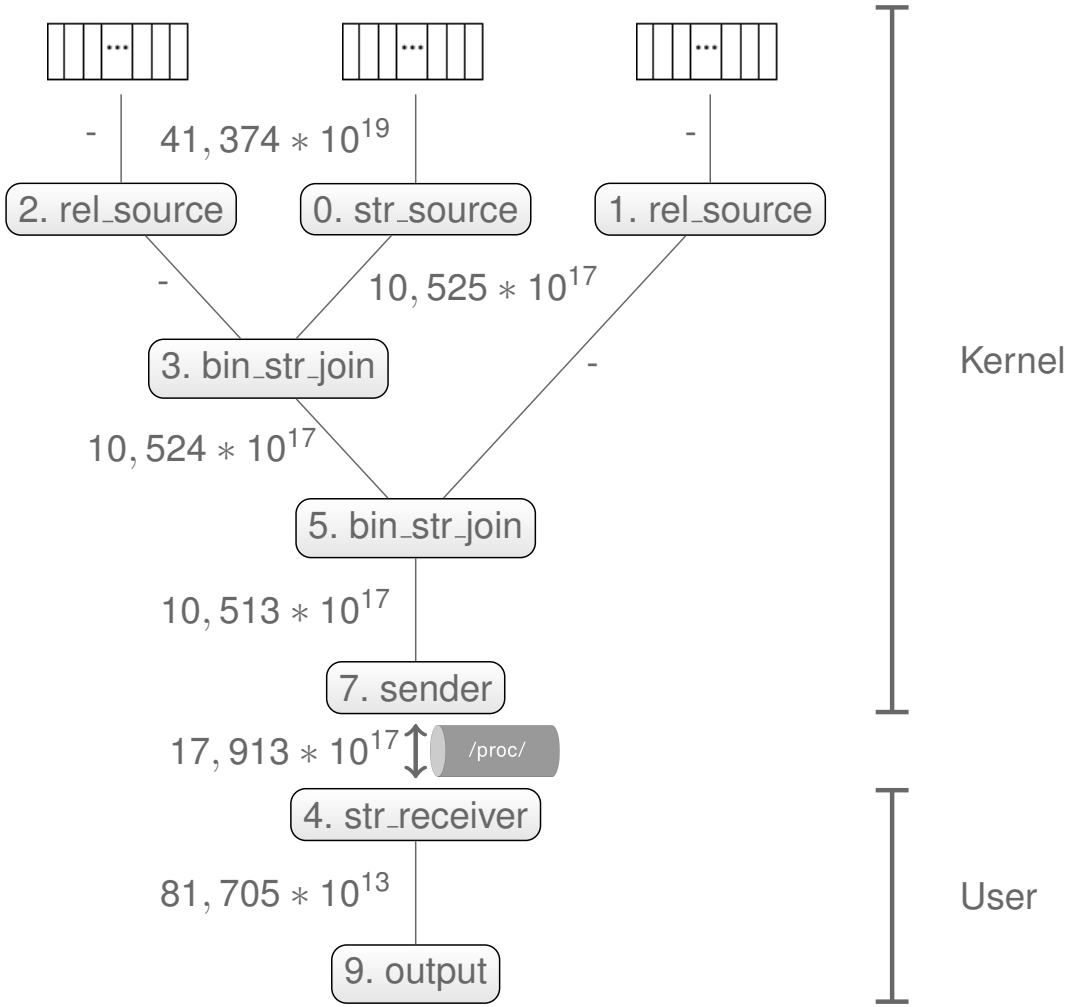
ESS

Transportzeiten in Nanosekunden (100000Hz)



technische universität dortmund

Streuung (100000Hz)



technische universität dortmund

Verbesserungspotential

- Die **Transportzeit** zwischen den Operatoren und zwischen den Puffern sollten optimiert werden.

technische universität dortmund

Verbesserungspotential

- Die **Transportzeit** zwischen den Operatoren und zwischen den Puffern sollten optimiert werden.
  - Durch eine Veränderung des Schedulingverfahren erreichbar.

technische universität
dortmund

Abschluss

Reduzierung der Systemlast

technische universität
dortmund

Abschluss

Reduzierung der Systemlast

- Verbesserung/Vermeidung der Funktionen ***getGlobalTS***, ***service_schedule*** und ***waitScheduler***

technische universität
dortmund

Abschluss

Reduzierung der Systemlast

- Verbesserung/Vermeidung der Funktionen ***getGlobalTS***, ***service_schedule*** und ***waitScheduler***

Verbesserung der Verarbeitungsgeschwindigkeit

technische universität
dortmund

Abschluss

Reduzierung der Systemlast

- Verbesserung/Vermeidung der Funktionen ***getGlobalTS***, ***service_schedule*** und ***waitScheduler***

Verbesserung der Verarbeitungsgeschwindigkeit

- Die **Transportzeit** zwischen den Operatoren und zwischen den Puffern sollten optimiert werden.

technische universität
dortmund

Abschluss

Reduzierung der Systemlast

- Verbesserung/Vermeidung der Funktionen ***getGlobalTS***, ***service_schedule*** und ***waitScheduler***

Verbesserung der Verarbeitungsgeschwindigkeit

- Die **Transportzeit** zwischen den Operatoren und zwischen den Puffern sollten optimiert werden.
  - Durch eine Veränderung des Schedulingverfahren erreichbar.

technische universität
dortmund

Abschluss

Reduzierung der Systemlast

- Verbesserung/Vermeidung der Funktionen ***getGlobalTS***, ***service_schedule*** und ***waitScheduler***

Verbesserung der Verarbeitungsgeschwindigkeit

- Die **Transportzeit** zwischen den Operatoren und zwischen den Puffern sollten optimiert werden.
  - Durch eine Veränderung des Schedulingverfahren erreichbar.

Ausblick auf weitere Untersuchungen

technische universität
dortmund

Abschluss

Reduzierung der Systemlast

- Verbesserung/Vermeidung der Funktionen ***getGlobalTS***, ***service_schedule*** und ***waitScheduler***

Verbesserung der Verarbeitungsgeschwindigkeit

- Die **Transportzeit** zwischen den Operatoren und zwischen den Puffern sollten optimiert werden.
  - Durch eine Veränderung des Schedulingverfahren erreichbar.

Ausblick auf weitere Untersuchungen

- Erschöpfende Analyse.

Abschluss

Reduzierung der Systemlast

- Verbesserung/Vermeidung der Funktionen ***getGlobalTS***, ***service_schedule*** und ***waitScheduler***

Verbesserung der Verarbeitungsgeschwindigkeit

- Die **Transportzeit** zwischen den Operatoren und zwischen den Puffern sollten optimiert werden.
  - Durch eine Veränderung des Schedulingverfahren erreichbar.

Ausblick auf weitere Untersuchungen

- Erschöpfende Analyse.
- Analyse und Bewertung der gewählten Messverfahren.

technische universität
dortmund

ESS

Abschluss

Reduzierung der Systemlast

- Verbesserung/Vermeidung der Funktionen ***getGlobalTS***, ***service_schedule*** und ***waitScheduler***

Verbesserung der Verarbeitungsgeschwindigkeit

- Die **Transportzeit** zwischen den Operatoren und zwischen den Puffern sollten optimiert werden.
  - Durch eine Veränderung des Schedulingverfahren erreichbar.

Ausblick auf weitere Untersuchungen

- Erschöpfende Analyse.
- Analyse und Bewertung der gewählten Messverfahren.
- Untersuchung anderer kCQL-Anfragen.

technische universität dortmund

Abschluss

Reduzierung der Systemlast

- Verbesserung/Vermeidung der Funktionen ***getGlobalTS***, ***service_schedule*** und ***waitScheduler***

Verbesserung der Verarbeitungsgeschwindigkeit

- Die **Transportzeit** zwischen den Operatoren und zwischen den Puffern sollten optimiert werden.
  - Durch eine Veränderung des Schedulingverfahren erreichbar.

Ausblick auf weitere Untersuchungen

- Erschöpfende Analyse.
- Analyse und Bewertung der gewählten Messverfahren.
- Untersuchung anderer kCQL-Anfragen.
- ...

Jochen Streicher, Alexander Lochmann und Olaf Spinczyk. “kCQL: Declarative Stream-based Acquisition and Processing of Diagnostic OS Data”. Englisch. In: *Proceedings of the Conference on Timely Results in Operating Systems (TRIOS)*. ACM, 2015. ISBN: 978-1-4503-3941-4/15/10. DOI: 10.1145/2834590.2834598.