A Qualitative Interview Study of Distributed Tracing Visualisation: A Characterisation of Challenges and Opportunities

Thomas Davidson, *Student Member, IEEE,* Emily Wall and Jonathan Mace

Abstract—Distributed tracing tools have emerged in recent years to enable operators of modern internet applications to troubleshoot cross-component problems in deployed applications. Due to the rich, detailed diagnostic data captured by distributed tracing tools, effectively presenting this data is important. However, use of visualisation to enable sensemaking of this complex data in distributed tracing tools has received relatively little attention. Consequently, operators struggle to make effective use of existing tools. In this paper we present the first characterisation of distributed tracing visualisation through a qualitative interview study with six practitioners from two large internet companies. Across two rounds of 1-on-1 interviews we use grounded theory coding to establish users, extract concrete use cases and identify shortcomings of existing distributed tracing tools. We derive guidelines for development of future distributed tracing tools and expose several open research problems that have wide reaching implications for visualisation research and other domains.

Index Terms—Visualisation, Distributed Tracing, Systems.

1 Introduction

Modern internet applications such as social networks and e-commerce sites are large-scale and distributed; their application logic comprises many inter-operating services deployed across different physical machines, all communicating over the network. The term *microservices* is often used to denote this architectural style of building applications. After these applications are deployed, developers and operators can make use of *Distributed Tracing tools* (**DT**) to observe, record, and troubleshoot the application's ongoing end-to-end behavior [1]. Distributed tracing emerged over the past decade in tandem with modern distributed application designs. Today, distributed tracing is gaining maturity, with recent open-source frameworks [2], [3], interfaces [4], standardisation [5] and bespoke solutions from startup companies [6], [7].

DT tools record rich, detailed execution traces of *requests* executed by the application – e.g. search queries, web page loads, or other top-level user interactions. Each trace portrays the end-to-end execution of a request, combining data from many services and machines. A single trace can be complex, since modern applications are large in scale and distributed across many servers, with dynamic topologies. Each request will involve many service calls across many machines; e.g., loading the home feed of a social media application will entail an execution spanning social graph databases, user and authentication services, advertising backends, and many more [8]. As such, effective use of this large and complex data presents a number of unique challenges in visualisation research.

Application developers, system operators, and site reliability engineers use DT to understand and troubleshoot deployed applications. They do so by querying and visually inspecting large, continually-generated datasets of traces [9]; system operators thus need effective visualisations to efficiently troubleshoot large and complex trace data. However, DT research to date has focused on the technical infrastructure for capturing trace data with relatively little focus on the end-user requirements of DT. Visualisations present in today's DT tools and published research literature were often developed for convenience and it is difficult to ascertain which, if any, aspects of their design were user-driven. Simultaneously, users of existing open-source and proprietary DT tools often struggle to work effectively and cite problems with usability or the design of the data visualisation itself (cf. §2.3). We posit that there is a critical missing piece hindering adoption of existing solutions: namely, foundational work toward understanding the challenges and potential benefits in the relatively nascent use of visualisation solutions in distributed tracing.

In this paper we present a qualitative interview study with six practitioners of DT across two large internet companies. Our study includes prominent users and maintainers of multiple different DT tools, both open-source and bespoke. We conducted two rounds of 1-on-1 interviews, establishing the setting of our users, discussing the problem area, and observing their interactions with existing tooling. From these interviews, our qualitative analysis identifies key use cases and challenges for DT tools. Finally, we synthesise eight guidelines for the design of DT. These outcomes provide the first systematic characterisation of this rich problem area for visualisation research. The guidelines (i) provide a framework for future development of DT analysis tools; (ii) identify opportunities for applying established techniques from visualisation literature to DT; and (iii) expose open problem areas, beyond the domain of DT, which visualisation could play a key role in solving.

2 Background

2.1 Motivation

DT is primarily used to monitor and troubleshoot a deployed application's runtime behavior. For example, a system operator might ask questions like "*why was this search query slow?*" Answering such questions is deceptively difficult because modern applications are typically built as distributed systems of inter-operating microservices. One request (e.g., the search query) has

- *T. Davidson and J. Mace are with the Max Planck Institute for Software Systems, University of Saarland, Saarbrücken, Germany E-mail: tdavidso@mpi-sws.org*
- *E. Wall is with Emory University, Atlanta, GA, USA*

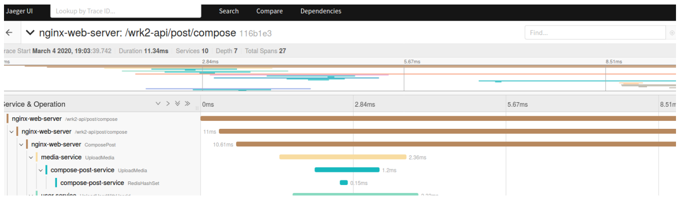
Jaeger UI
Lookup by Trace ID...
Search
Compare
Dependencies
nginx-web-server: /wrk2-api/post/compose 116b1e3
Find...
race Start March 4 2020, 19:03:39.742
Duration 11.34ms
Services 10
Depth 7
Total Spans 27
ervice & Operation
0ms
2.84ms
5.67ms
8.51m
nginx-web-server /wrk2-api/post/compose
nginx-web-server /wrk2-api/post/compose
11ms
nginx-web-server ComposePost
10.61ms
media-service UploadMedia
2.36ms
compose-post-service UploadMedia
1.2ms
compose-post-service RedisHashSet
0.15ms

Fig. 1. Trace visualisation from Jaeger [2], an open-source DT tool. The trace depicts a `ComposePost` API call to the DeathStar social network application [8], a small-scale open-source microservice benchmark.

an end-to-end execution that spans many different services and machines. Each service is only responsible for a small piece of request logic, and a problem's root cause (e.g. an overloaded backend database) could be in any of those services or machines.

DT records *end-to-end* data about request executions. DT is related to traditional event logging techniques, but with three important distinctions: (1) it coherently collects and combines all events logged by a request *across all machines visited*; (2) it distinguishes and separates events generated by different concurrent requests executing at the same time; and (3) it explicitly records the causal ordering of events. For each traced request, DT ultimately captures a richly annotated graph of events spanning all visited machines. DT is useful for many troubleshooting tasks that are otherwise difficult or impossible; users can inspect and query traces to observe the full end-to-end flow of any request's execution.

Human users (application developers and operators) consume trace data via frontend visualisations and UIs. However, since DT is still nascent, research has not focused on user-driven development and design. Instead, prior work from industry [9], [10], academia [11], and open-source [2], [4] has, necessarily, first tackled the technical challenges of recording and constructing traces from data that is scattered and interleaved across many machines. Visualisation has begun to receive attention in recent work, albeit still with a focus on tracing mechanisms and absent of user-informed design aspects [12]. Outside of academia, several recent DT startups have extended open-source solutions such as OpenTelemetry [4] and Jaeger [2] but we find no published design study work or novel visualisation approaches.

To date, there exists no reference point for user requirements or preferences in DT. Our goal in this work is therefore to provide scaffolding to inform future visualisation solutions by providing a rigorous accounting of users, tasks, and challenges involved in DT, as well as opportunities for visualisation innovation.

2.2 Distributed Tracing Overview

We introduce the key concepts of DT with reference to Fig. 1, which illustrates a "swimlane" visualisation ubiquitous in DT.

Where does trace data come from? When a developer wishes to use DT in their application, they link a DT library and use an application-level tracing API to record information. The DT library will record and transmit the logged events; DT backends will receive and combine events into full request traces, and store them in a database. Developers and system operators can subsequently access traces via the web UI of the framework.

Trace. One trace represents one end-to-end request. Fig. 1 illustrates a trace of a "`ComposePost`" request in the DeathStar social network application [8], a small-scale open-source microservice benchmark. Each trace is a directed, acyclic graph (DAG) of *spans* (vertices) and *relationships* (edges).

Spans. Spans are the building blocks of a trace: a span represents a segment of processing that occurred, such as a single function execution, a single thread execution, or a remote procedure call (RPC). The granularity of a span is chosen by the software developer when using the tracing API. Spans include timing information, and the developer can attach arbitrary key-value *attributes* using the tracing API (e.g. "`span_name:UploadMedia`"). The horizontal bars in Fig. 1 depict spans, their latency, and the "`span_name`" attribute. Developers can *annotate* spans with timestamped log messages (not depicted in Fig. 1). The information in a span is developer-provided and thus varies between different applications or even services within the same application.

Relationships. Traces also capture *relationships* between spans, which typically correspond to inter-thread or inter-service communication. For example, RPC communication is recorded as a *parent-child* relationship between the caller and callee spans. Fig. 1 depicts parent-child relationships using indentation and nesting. Spans can have arbitrary relationships to other spans, but parent-child relationships are most common in RPC-based applications. Like spans, relationships can have additional attributes and annotations.

2.3 Visualisation

The following descriptions do not refer to a single visualisation, but are common features across most state-of-the-art solutions.

Accessing Traces. Traces are hosted in a backend database and exposed via a web interface. Users (i.e., developers and operators) can find traces by searching on high-level, common attributes such as a specific service, API call type, or time window. From the list of matches, users can select an individual trace for closer inspection.

Swimlane View. The swimlane – also referred to as waterfall – visualisation is the canonical way to visualise individual traces used by prior DT frameworks [9], [10]. It enables users to manually investigate an individual request in detail as part of a troubleshooting task. Fig. 1 shows a representative example of a trace visualisation in Jaeger, the state-of-the-art open-source DT framework. The trace depicts a timeline of a single request, with spans depicted horizontally and sorted vertically. Some visualisations explicitly depict relationships as lines between spans [13]; however in Fig. 1 relationships are implicit based on vertical ordering and indentation of span names. Additional information such as annotations and key-value attributes of spans are typically available by clicking on a span (through a dialog box or side panel). An individual trace can be large (thousands of spans; several MB in size) so there is typically some form of pan-zoom navigation. Although swimlane visualisations are widespread in practice, there is no known rigorous justification for their design, and users often encounter difficulties, e.g. missing critical features, inconsistencies in presentation, time-consuming and confusing manual steps, and more [14], [15].

Alternate Views. Some DT tools do offer alternatives to the swimlane view, such as Jaeger's service dependency view, and many tools offer multiple subtly different approaches to view the same trace, such as SkyWalking's List, Tree, and Table views. However, every tool that we examined (whose capabilities are covered in Table1) used the traditional swimlane view. From our interviews and analysis of features, the swimlane view appears to be the most pervasive and so we focus on this.

Aggregate visualisations. DT can capture large volumes of traces: Facebook reports over 1 billion traces per day [9]. Overall, the collection of request traces represents the system's behavior as a whole. Some recent systems provide visualisations for aggregate analysis. For example, Canopy extracts per-trace metrics to a tabular form, which operators can then query using standard time-series database interfaces [9] (e.g. "what is the latency distribution of the `url-shorten-service` service"). Jaeger provides an experimental trace comparison feature to compare the structure of a trace to a set of other traces [16]. Like the swimlane view, design choices and alternatives are not discussed in these prior works.

2.4 Related Work

DT shares similarities with program and software tracing that have also utilised visualisation [17], [18] including work in single-machine application tracing which utilises a similar swim lane approach [19]. However, several aspects of DT make it significantly different from these approaches:

Distributed. DT combines trace events from multiple machines, must correctly distinguish concurrent requests, and must coherently order events. Existing single-machine tracing techniques do not trivially extend to a distributed setting.

Ad-hoc Data. DT data is generated at the application level and there are few guarantees about the structure, contents, or detail of trace data. Different developers and application components can choose to record different things.

No model or schema. The application's workload and topology are dynamic and change often. The DT user often does not have any access to source code or data schema, and there is no reliable model of the system's components and dependencies. Conversely, traditional approaches deal with individual runs of binaries in offline settings, often with access to source code.

Datasets of traces. DT records *many* traces of *many* requests; by contrast traditional tracing approaches are concerned with individual program runs in isolation. DT aims to connect and compare individual trace executions to this aggregate context.

Online setting. DT targets large always-on systems; traces are continually generated; and users care about trends and changes over time. By contrast traditional approaches typically trace individual runs of binaries in an offline setting.

Timing. With no global clock, DT orders events using the "happens-before" relation [20]. Single program traces are less ambiguous due to accurate timing from a shared clock.

Use Cases. Survey work from 2014 found that traditional program tracing use cases could be simplified to: global comprehension, problem detection, and diagnosis and attribution [21]. These are aligned to use cases we establish later (§5) but have important differences. Single program tracing is concerned with drilling down to the instruction or function level, and debugging programs in standalone environments. DT has a much coarser granularity and is concerned with the macro level such as identifying the service where a problem is occurring and contacting the service owner. DT is also used much more proactively to pre-empt bottlenecks and mitigate them, rather than reacting to exhibited problems.

Prior works often have a specific use case focus, e.g. detecting latency [22] or debugging [18]. Conversely, our work lays foundations to establish the problem area of DT, why visualisation is necessary for it, why existing solutions fail to satisfy the use cases, and steps that future work can take to mitigate this.

Despite these differences, work in DT visualisation can still be informed by this work, e.g., in areas such as visualising parallel traces which occur concurrently [22] and the use of adaptive timescales [17]. Most work in this area introduces solutions and evaluates their efficacy [22]; however, what this prior work collectively lacks is a systematic and in-depth understanding of the fundamental nature of the problems. Therefore, instead of introducing another solution for DT, we first provide a rich characterisation of the domain and associated problems.

Performance visualisations are also widely used in other High Performance Computing domains [23], with some work focusing explicitly on user requirements [24]. Other performance and troubleshooting scenarios have considered visualisations such as flame graphs [25] for resource consumption. Dashboards for system monitoring [26] have also utilised visualisation in detecting and analysing anomalous performance in cloud systems. Contributions from these areas may be relevant, but there are fundamental differences and users have different tasks and goals. A goal of our study is to solidify these goals in the DT domain and extract a characterisation highlighting the unique challenges DT presents.

Throughout this paper we identify several avenues of visualisation research that can be applied to DT. This includes work on optimal graphical encoding channels for various data types [27]; visualisation design mantras such as "overview first, zoom and filter, then details on demand" [28] or focus+context [29]; evaluation methodologies [30], [31] ; and lessons learned from applications for program understanding and debugging [32]. Future work can also be influenced by work investigating the use of guidance in visual analytics [33] and the difficulty of designing visualisations for specialist domains [34] – specifically how designers can collaborate with domain experts to efficiently design practical and useful tools.

3 Interview Study

Qualitative interview studies are useful for understanding and characterising a problem area, especially from an end-user perspective. Interview studies have helped in advancing diverse fields such as data journalism [35] and requirements engineering [36]. Adjacent to DT, interview studies of data analysis tasks [37], [38] have had wide-reaching impact on the subsequent development of data analysis tools [39], [40] ; likewise studies of dashboard design [41], [42] laid the foundations of later visualisation designs [43]. Correspondingly, we hope to influence future work on DT visualisation, as no prior characterisations exist of this problem area.

In this section, we describe a qualitative interview study performed with six DT practitioners from two large internet companies. Our goal is to characterise the problem area by: (a) establishing the common user roles of DT visualisations (cf. §4); (b) establishing typical use cases and workflows (cf. §5); (c) identifying challenges with existing tooling and where it does not meet users' needs (cf. §6); and (d) synthesising guidelines for the development of future distributed tracing visualisation solutions (cf. §7).

3.1 Participants and Setting

DT is nascent, and consequently widespread adoption and understanding of DT analytics is still in its infancy. Yet, in order for us to understand the existing problems, it is crucial to work with highly experienced participants. We sought users with experience of popular open source tooling. Furthermore, seeing how companies adopted these open source solutions into bespoke visualisations enabled us to learn from their approaches. The resulting pool of potential participants was consequently small, when factoring in our other inclusion criteria.

We used network and reputation-case selection [44] to contact two large internet companies. One company exclusively used open source solutions and the other had developed an in-house bespoke DT solution over several years. Initially, we sought to recruit users of DT solutions who were also involved in its maintenance: a backend and frontend developer of the tracing systems at each company. Taking inspiration from the winnow and cast stage (solidifying collaborators and roles) of the design study framework [45] we expanded this to include a user from each company who exclusively consumed visualisations and was not involved in maintenance. This resulted in a total of six participants (one maintainer, developer, and user from each company) who each had 5+ years of experience with DT. The participants were:

P1 Technical lead overseeing DT projects at company A, including backend and frontend development. Active maintainer of a leading open source tracing solution.
P2 Technical lead of performance and observability visualisation at company A, primarily focused on frontend development.
P3 Data scientist working on prelaunch product at company A, daily workflow heavily involves DT.
P4 Visualisation designer on the performance team at company B, focused on frontend development.
P5 Senior engineer responsible for DT infrastructure at company B, including backend and frontend development, and maintainer of a leading open source tracing solution.
P6 Senior Software Reliability Engineer for major product at company B, daily workflow significantly incorporates DT.

All interviews took place using a video conference system. Where appropriate we asked participants to share their screens. All participants spoke with us from a typical working location.

3.2 Method

We follow established guidelines for planning and executing qualitative interview studies [44] and adopt approaches used for interview studies in similar fields [37], [38]. We designed an interview guide [44] to ensure each participant covered a core set of questions; otherwise, interviews were semi-structured [46], [47] to allow participants to highlight what they thought was important, and to allow broad coverage of the problem area. The first round aimed to characterise DT tools, users, and use cases. For the second round we observed each participant using their existing tools and asked for explanations of their workflows and to highlight problem areas. Each interview lasted between 1 to 2 hours and was video and audio recorded and transcribed by the lead author.

We applied a grounded theory approach [48] to extract salient codes from each interview. The lead author then applied an open coding approach to the interview transcriptions [49]. The codes were iteratively reviewed by the second and third authors, and collectively discussed until a consensus was reached and the final codes were compiled. The lead author then applied affinity diagramming to form broader categories of codes, collecting and adapting the codes across all participants into coherent subsets.

To seek saturation of codes and mitigate the small size of our user pool, we revisited several themes raised by specific participants in subsequent interviews with the other interviewees.[1] We repeatedly discussed and refined our categorisations and classifications based on coding calibration techniques described in prior research [50]. After four iterations we reached a set of stable codes for the users and use cases after our first set of interviews.

1. We include anonymised supplementary material to further detail the coding process and iterations

During the first interviews, some codes emerged relating to tooling-specific commentary. As it was not the focus the saturation of these was unclear. Therefore, during the second interview we explored how users interacted with their existing tools, and the problems that arose. Following this round we followed the same coding methodology from round one and reached a set of stable codes for existing challenges.

From these three sets of codes (users, use cases, and challenges), we proceeded to identify areas for potential improvement where challenges could be addressed. The lead author drafted broad areas for improvement and after several iterations with co-authors we arrived at a set of improvement areas which formed the basis of our design guidelines. To ensure a comprehensive accounting of our findings, we utilised a framework to map each improvement areas to a challenge. We adapted the improvement areas to extract eight guidelines for design which ultimately gave us full coverage of our extracted challenges that users faced frequently.

4 User Roles

In the first interview round we asked our participants to classify the user groups involved with DT. Participants consistently identified two main groups: **developers** and **consumers**. Our participants P1, P2, P4, and P5 represent this first group and participants P3 and P6 the latter. All 6 of the participants identified this divide.

Developers maintain the tools themselves and also frequently interact with frontends. Developers subdivide into these three categories: backend engineers, Software Development Kit (SDK) engineers, and visualisation / frontend engineers. Backend engineers focus on the implementation of the DT instrumentation. SDK engineers were primarily considered to be responsible for backend focused work, such as sampling policy. Visualisation and frontend engineers develop the existing frontend tooling that the companies used, or tweak open-source implementations. Developers have an in-depth understanding of the functionalities and data available with DT, as they are involved in implementing the tooling themselves.

Consumers make use of trace data for troubleshooting and analysis tasks. Consumers primarily interact with the DT frontends and do not require knowledge of DT internals. Consumers include developers of individual services within an application; developers responsible for the underlying infrastructure; and site reliability engineers responsible for application reliability end-to-end. The most notable finding concerning consumers was that they are isolated from one-another. An organisation as a whole may have many DT consumers, but in any given developer team there would typically only be one engineer who was an expert in DT and would use it extensively; other engineers on the team use it occasionally, or not at all. We found that these expert DT consumers had minimal communication across teams and as a result had little insight about how other consumers were using the tools available.

5 Use Cases

We summarise the use cases in five main categories, plus a handful of less common use cases that could not be clearly categorised.

Incident localisation. Troubleshooting arose as the most prevalent use case from all interviews. Participants often described having to localise incidents and using DT to narrow down where an actual problem had occurred. The purpose of this localisation is to identify the service containing the root cause, and by extension the correct engineering team to contact – DT is *not* heavily used to subsequently identify the location of errors in the code.

Application behaviour in aggregate. Every participant described a workflow starting from aggregate metrics before narrowing down to specific traces to inspect. Moreover, many investigations do not involve interacting with individual trace visualisations in any form; in this case users work only with recorded metrics and aggregated data over sets of traces. Understanding latency distributions of traces and spans emerged as a prevalent use case.

Proactive measures. DT data is heavily leveraged for proactive measures such as resource and capacity planning. Some proactive measures are necessary due to technical limitations of the tracing system. For example, reactive troubleshooting is impractical if it requires recording new information as it can take hours to days for changes to be incorporated and new data ready to be queried.

Network connectivity. Users want to leverage the causal nature of DT to understand the connectivity of services within their application. Two specific use cases emerged. First, understanding how a service connects to other services in its immediate neighbourhood – who it calls and who calls it. Second, understanding the indirect impact of service calls on downstream services and how upstream services dictate the number of times their service was called.

Trace inspection and hypothesis validation and generation. Users often seek exemplar traces for validating hypotheses. Typically, users arrive at individual traces only after utilising other tools or manually querying trace data. Users generate their hypotheses using these other tools then navigate to individual traces to validate that hypothesis. This is a two-way process, with inspection often leading to new hypotheses and revisiting of other tools. As part of this workflow users commonly compare two or more traces.

Further specific use cases. Participants described a handful of other use cases. One of significant note was the use of DT to analyse timing and latency details more closely. This could involve looking at latency distributions, but mainly using the richness of DT data to see timing details per span (often manually querying the data) and to troubleshoot when discrepancies in timings arose.

6 Challenges

We describe the challenges that users experienced in five primary categories that are labelled Challenge (**C**) **1** - **5** for future reference. This list is not intended to be exhaustive, but rather give a sense of (potentially overlapping) challenges that users experience.

6.1 C1: Mismatched Workflow

Individual trace visualisation is only one piece of the troubleshooting workflow. All participants described workflows beginning from high level metrics and aggregated overview data (tabular datasets derived from traces) and leading eventually to inspecting individual traces of interest.

Investigative processes thus often span multiple UIs and tools and feature repeated use of the same visualisations for different source data. Currently, only a very narrow portion of a task may be performed within a single visualisation or UI; users often utilise multiple different UIs and different instances of specific visualisations when performing an analysis.

> "*This tool is a remarkably good and dense visualisation but it is at a certain high level of abstraction, so you'd want to couple it together with other tools.*" – **P4**

This is not a one-way workflow as often users need to traverse back up the tree and move to other tools to continue their investigations. Tasks often involved users forming and validating hypotheses, and this necessarily requires a lot of exploration and backtracking, both within and across tools.

Tools are myopic. Despite an ecosystem of tools and workflows spanning multiple tools, by default each tool is presented as a standalone option to its users. Participants describe how visualisations attempt to be 'one size fits all' solutions despite being rarely used in this manner. This results in a disparate toolset and burdens users.

Linking between tools is not supported. Despite a common progression from overview down to trace inspection, tools are not coupled and must be manually accessed by users, navigating to the relevant data from scratch each time. All participants expressed a desire for linking between tools for more direct navigation and frustration at siloed tools slowing down troubleshooting. In §8 we discuss industry efforts to address myopic tools and their linking.

Missing broader context. Individual trace inspection is typically just one step in a broader troubleshooting workflow. For example, a user might inspect a trace to determine if it exemplifies some performance anomaly; the broader context of *what* the anomaly is derives from prior steps in the workflow. Participants described how they do not gain significant benefit from any one tool or visualisation in isolation. However, existing tools only present trace data in a generic way, regardless of how or why a user arrived at the tool. For example, observing an individual trace and its relative timings is not valuable unless you know whether these timings are slower or faster compared to the general corpus of traces.

6.2 C2: Excessive User Burden.

DT supports diverse user groups, a wide range of use cases, and each investigation can be slightly different. Likewise, traces are a rich data source with a large number of attributes attached to every aspect of a trace; for example, every span will have multiple key-value pairs and annotations. Consequently there are a multitude of different ways to visualise traces. Existing tools attempt to satisfy the broad range of use cases, yet offer little customisation, resulting in cluttered visualisations containing frequently unnecessary and distracting information for users.

Irrelevant data. Users have different requirements about which data they want to see and how they interact with the visualisations. This encumbers experienced users; most participants expressed frustration at the way information was displayed to them and discussed having to learn to just ignore aspects of it.

Experienced users are distracted by the information irrelevant to their task and would rather narrow down to specific pieces. When demonstrating existing tools, several participants remarked that they don't care about most of the data displayed and wish they could exclude it completely.

Participants also described how novice users suffer from information overload. For new users, trying to display all of the pertinent information is giving too much information at once.

Useful information is often hidden. Several participants expressed frustration that information would be buried beneath several clicks, slowing down investigations unnecessarily. In some cases, visualisations would completely omit data from a trace that users knew was present, forcing them to seek it elsewhere.

> "*I need to look at the JSON for this trace to actually see what resolution these timestamps are actually in.*" – **P5**

Insufficient customisation. Some tools lack customisation clearly desired by users. The idea of subtly different data presentations facilitating new discoveries was present in our interviews, reflecting results in literature such as the 'pop-out' phenomenon [51] and different visualisations eliciting different results [52].

"*For a lot of people the visual aspect, having that pop out and shifting how it appears is very useful.*" – **P6**

Repeated effort. Users often repeat the same actions many times, both as part of distinct workflows and within the same workflow. Investigative processes often feature repeated use of the same visualisations but for different source data (e.g. opening multiple tabs to look at different exemplar traces). All participants highlighted how repeated actions and manual effort can significantly slow down investigations. In some situations this was so pronounced that our users wouldn't carry out what they knew would be useful analysis because of the time and effort requirements.

"*I would say the number one reason is that I just don't have time. I would love to be able to explore some of this more... things I'd like to build and questions I would like to answer which I can't at the moment*" – **P6**

For example, users will typically arrive at a UI midway through an investigation (**C1**) with an idea of what they are looking for. The tools we observed with our end users often present traces with minimal interaction support requiring the user to then visually scan to find relevant details. This is a problem we see echoed in other industry efforts too (Table1). In several of the demonstrated tools, users must perform ad-hoc visual explorations repeatedly with multiple different traces to locate details. None of the tools we observed allowed users to persist or re-use state.

Mental model. Users require a thorough mental model to interpret a trace they are shown. Users often investigate things that 'don't look right', drawing from their understanding of 'normal' system behavior. This is feasible for experienced users but greatly increases the barrier to entry of this form of analysis to any new users.

6.3 C3: Unsupported Interactions

Several interactions featured prominently in our interviews as important parts of troubleshooting workflows, yet are unsupported in existing tools or possible only with ad-hoc workarounds.

Programmatic access. Troubleshooting with DT is inherently open-ended, so inevitably users encounter a task that is not supported by the current tooling, such as performing customised aggregate analysis over large volumes of traces. In users' current workflows they instead must transition to progammatic APIs or console tools to load or query the underlying trace data. While programmatic access to traces is a necessary step, current tools provide no assistance. Initiating this transition is cumbersome and involves excessive manual effort to copy and paste trace IDs or write long SQL queries to obtain traces from backend storage.

Aggregate analysis. Aggregate analysis arose for a number of use cases yet is not supported in current tools beyond querying high-level tabular datasets (pre-defined features extracted from traces). Users must manually perform aggregate analysis using programmatic access, writing queries to load and process the trace data, and generating custom reports in analytic notebooks. Doing this for common tasks requires substantial repeated effort and is error prone, time-consuming, and difficult to repeat.

Navigating a trace. Existing trace visualisations assume that users are performing an open-ended manual exploration. However, following from **C1**, in practice users arrive at a visualisation with a specific hypothesis in mind to investigate. In these cases, exploration-based navigation is a poor fit for users, who instead would resort to ad-hoc workarounds such as using browser-based text search to identify spans with specific names.

In general, participants expressed a desire for searching on trace attributes and frustration with existing tools. Moreover, users need to search on *any* elements of the trace data – even if they are not explicitly displayed in the current visualisation – since the user might have arrived via a different visualisation that *does* display the data, or the attributes might be hidden in the current view.

Lastly, users often want to undo or rewind actions, yet this is unsupported in existing tools. Instead, users sometimes would have to restart from the beginning.

"*Remove filter just always pops you all the way back up to the top, it doesn't keep the stack.* " – **P4**

Trace Comparison. All participants described comparing one trace to another as an interaction modality. No existing trace visualisation system facilitates in a meaningful way comparison of one trace to another(Table1). All users described cumbersome approaches to comparing two traces which effectively boiled down to opening up multiple browser windows and comparing visually:

6.4 C4: One Size Doesn't Fit All

There is no consensus on the best way to visualise individual traces, and it is unlikely that one visualisation is appropriate for all tasks due to the open-ended nature of troubleshooting. However, participants described how development efforts not only focused on a 'best' visualisation approach, but also actively pushed deprecating existing approaches that users liked. For example, a developer describing another participant's primarily used vis:

"*Yeah so [old vis] is horrendously bad. We want to basically stop maintaining and deprecate it*" – **P2**

Surprisingly, we learned that developers of in-house tooling currently receive minimal feedback on which aspects of the tooling are frequently used or what could be improved. This may arise in part due to tracing users being relatively disconnected (cf. §4). We believe this problem is further exacerbated for open-source tools, whose users are even further disconnected from developers.

6.5 C5: Data Quality

We identify several problems related to data quality. One was simply linked to the longevity of trace data – tracing backends only keep data for one or two weeks – making it difficult to reproduce analyses. In addition, there were problems related to malformed data: how it was presented in the UI and the impact that sampling policy could have on the truthfulness of data being presented.

Tools focus on the 'ideal' trace. Existing tools are designed to visualise 'golden case' trace data. However, troubleshooting implicitly means looking at traces that might deviate from normal to try to understand a problem. For example, troubleshooting long-running requests inherently requires visualising very large traces, which some tracing visualisations could not do.

"*I know there's no point trying to open something that's too large in the UI*" – **P5**

Data can be missing or malformed. Users found it difficult to ascertain if data was malformed or there was a problem with the UI. Without even knowing for sure if the problem is in the UI or the data this often leads to users simply having to throw away the trace despite it potentially holding valuable insights.

TABLE 1
Capabilities of Existing Tools

	Comp. A	Comp. B	Jaeger	Zipkin	Lightstep	X-Ray	Datadog	Elastic	Honeycomb	New Relic	Signoz	SkyWalking	DynaTrace	Sentry
Is the tool open source?	☒	🟩	🟩	🟩	☒	☒	☒	☒	☒	☒	🟩	🟩	☒	☒
Does the tool use swimlane view for individual traces?	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Can you view an individual trace in an alternative way?	🟩	🟩	🟩	☒	☒	☒	🟩	☒	☒	☒	☒	🟩	▲	☒
Can you customise the display of a trace? **(G1/2)**														
Change the colours?	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒
Expand/collapse spans?	🟩	🟩	🟩	🟩	🟩	🟩	☒	🟩	🟩	🟩	🟩	☒	🟩	🟩
Show/hide additional features of a span?	🟩	☒	☒	☒	☒	?	☒	☒	▲	🟩	☒	☒	?	?
Is the customisation persisted? **(G1)**	☒	☒	☒	☒	?	?	?	?	▲	☒	☒	☒	?	☒
Can you search within a trace? **(G3)**														
On service name?	🟩	🟩	🟩	☒	🟩	☒	?	☒	🟩	☒	☒	☒	🟩	🟩
On other features?	☒	☒	☒	☒	☒	☒	?	☒	☒	☒	☒	☒	🟩	?
Switching between tools facilitated? **(G4)**	🔗	☒	☒	☒	🔗	🔗	🔗	🔗	🔗	🔗	🔗	☒	🔗	🔗
Easy to drop to programmatic access? **(G4)**	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒
History of trace interactions available? **(G5)**	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒
Does the trace view indicate outlier traces? **(G6)**	☒	☒	☒	☒	☒	☒	☒	☒	🟩	?	☒	☒	☒	☒
Does the trace view indicate outlier spans? **(G6)**	☒	☒	☒	☒	☒	☒	☒	☒	🟩	?	☒	☒	☒	☒
Can you compare two individual traces? **(G7)**	▲	☒	▲	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒
Can you compare two sets of traces? **(G7)**	🟩	☒	☒	☒	☒	🟩	☒	?	🟩	?	?	☒	?	?
Can the tool display malformed traces? **(G8)**														
Does it flag errors?	🟩	☒	🟩	☒	🟩	?	🟩	🟩	🟩	🟩	🟩	☒	🟩	🟩
Can it display a trace if it is missing data?	☒	☒	🟩	☒	☒	?	?	?	?	🟩	?	?	?	🟩
Can the tool display large traces? **(G8)**														
Overview display of the whole trace?	☒	☒	🟩	☒	🟩	☒	☒	☒	☒	☒	🟩	☒	☒	🟩
Can the UI load large (> 1000 spans) traces	🟩	☒	🟩	☒	?	?	?	?	?	?	?	?	?	?

🟩 Yes ☒ No ▲ Tool partially satisfies the question ? Not possible to ascertain 🔗 Tool satisfies the question, but only within its own infrastructure

7 Guidelines and Research Opportunities

We propose eight guidelines for the future design of DT visualisation and clarify their impact on further research. The following table summarises how each of these guidelines addresses one or more of the five challenges from §6.

	G1	G2	G3	G4	G5	G6	G7	G8
C1 – Mismatched workflow	✓	✓	✓	✓	✓			
C2 – Excessive user burden	✓	✓	✓	✓	✓	✓	✓	✓
C3 – Unsupported interactions			✓	✓		✓	✓	
C4 – One Size Doesn't Fit All				✓	✓			
C5 – Data quality								✓

These guidelines are intended to provide direction for developers of future tools for DT analysis, whilst also highlighting areas of visualisation that can help mitigate the identified problems and benefit from further research in this domain.

7.1 G1: Customisation and Persistence (C1, C2)

Our first guideline advocates that customisation should be a high-priority feature for DT visualisations.

Prioritise customisation Customisation is desirable as there are many diverse users and power users. We believe that the solution to this is to facilitate and prioritise customisation of the visualisation rather than trying to focus on one visualisation approach that satisfies all users and use cases. For example, users should be able to order key fields, change colour schemes, and alter the presence or absence of visualisation aspects (such as connecting lines).

Persist customisation Customisation should persist – users do not want to have to set up their ideal 'view' every time they come to inspect traces but should be able to save their preferences and select the configuration that best matches their investigation. Doing this reduces effort each time a user revisits a visualisation and streamlines the ability to repeat a workflow as they can jump directly to what they need.

Customisation and persistence offer a wealth of future potential research directions which we discuss in depth in §9. An interesting challenge with DT is to determine how and where to bind the persistence of customisation. One option is to bind decisions per user; another could facilitate team members operating with a shared persistence; another is to persist visualisation customisation elements per individual trace; and another is to persist visualisation customisation to a user session that spans multiple tools.

7.2 G2: Simple Starting Point (C1, C2)

While power users are a common user group of DT, it is also important to provide an accessible entry point for novice users. Hence we advocate for the initial view of a trace in tooling to be *as simple as possible*. A simple starting point will reduce the initial visual clutter that new users currently experience. That is, designers may consider a constructivist alternative to the common visualisation mantra: "overview first, zoom and filter, details on demand" [28]. Expert users may build a desired view based on their intuition and understanding of the data and system, expanding and investigating the relevant parts of the visualisation as necessary, building off of work on large graph exploration [53].

When coupled with further customisation and persistence (**G1**), users can simply return to this view whenever they want. Conversely, it also facilitates new users exploratively discovering aspects of the tracing visualisation without being overwhelmed by the initial view. One participant highlighted ongoing efforts to improve usability for novices:

"*Turning these overwhelming/complicated tools into something actionable to somebody that hasn't taken a class is a challenge we're starting to work on.*" – **P2**

The first two guidelines address **C1** by not restricting how a user can interact, and address **C2** by allowing users to control the amount of information they are presented with.

To implement this guideline, we can benefit from prior research on domain-specific adaptations of the information-seeking mantra [54] and work on *progressive disclosure* [55].

There is also an opportunity for further work to reduce the threshold of experience and knowledge required to make progress in DT through approaches such as mixed-initiative user interfaces [56] to explore the balance between human and machine effort – automating some things while leaving some judgement to users to create a 'best of both worlds' situation for human-machine teaming by leveraging automated analysis of the huge amount of data available and presenting it to users.

7.3 G3: Search-based Navigation (C1, C2, C3)

Search is a key interaction mode that differs significantly from manual exploration. Users often know what they're looking for – something specific in a trace. Search must therefore be a fundamental interaction supported by tools. Moreover, users need to search on all elements of the underlying trace data, even those not being displayed in the current visualisation. In support of search-based navigation, particularly for novice users, efforts may be informed by decades of HCI research on direct manipulation [57]. In particular, dynamic query widgets [58] are shown to improve both performance and user satisfaction for searching and filtering.

This guideline addresses **C3** by facilitating tasks that users have a desire to carry out but currently can not do easily with the existing tooling. Furthermore it addresses **C1** by allowing more of the data to be leveraged, rather than the existing narrow view of workflows and reduces the user burden from **C2** by allowing greater exposition of the underlying data.

7.4 G4: Interlinking (C1, C2, C3, C4)

Ideally, tools provide links to visualise data in other tools. This immediately reduces the amount of effort on users as they can navigate directly between tools and easily go back and forth. This guideline is not directly related to visual design but is nonetheless a critical part of the ecosystem of usage.

Loosely-coupled tools. While interlinking can be implemented by directly linking to other tools, this approach is brittle over time due to the continual development of new tools and interfaces. Eventually it will be infeasible to manually link each tool to each other tool. Some existing proprietary DT solutions – such as Lightstep [6] – have begun to address this problem by building 'comprehensive tools'. This approach does allow for easier linking between sections of the analysis pipeline, but it is unclear whether this approach will be sufficiently scalable to deal with new tools and visualisation approaches, or flexible enough to address the needs of users – such as our users who we found enjoyed the ability to choose between different tools for visualising traces.

Ideally, tools can make use of a level of indirection via some external management framework. Developers and designers of tools would only need to expose the relevant types of data they can visualise. This requires an infrastructural change that we strongly believe would aid both users and developers of the tools in more smoothly linking the different aspects of a DT workflow together.

Reusing state instead of starting from scratch. Users often want to take the state from one step of their troubleshooting and apply it to another, e.g., while looking at one trace, the user may want to then load different data. Systems should support using a current visualisation as context for populating other visualisations.

Dropping to programmatic access. Some of the problems that require users to go away and look at the underlying data could potentially be solved in situ by exposing more of the data. Tools should make it easy to take data and state and begin working programmatically. This can be achieved by exposing the specific queries to the underlying trace datastores that reproduce the data being visualised. We recommend that tools can ease the transition by providing prepopulated synthesised queries for similar traces that can be directly executed in a console. Moreover, due to the prevalence of browser-based analytics notebooks, tools could directly pre-populate a runnable notebook on behalf of the user, and open it in a new browser tab. This would substantially cut down time and errors introduced by doing these steps manually. There is minimal prior work into linking visualisations to programmatic data access, but some inspiration may be found in 'Utopia Documents' [59] which aimed to link academic articles with research data and has seen some use in bioinformatics fields.

Successfully implementing interlinking would greatly reduce the impact of **C1** and **C2** by reducing the existing extreme isolation of DT visualisation and instead beginning to recognise individual trace visualisation as one part of a rich analytical workflow. It additionally addresses **C3** and **C4** by making the transitions between tools less disruptive and reliant on existing knowledge.

7.5 G5: Interaction History (C1, C2, C4)

History within a visualisation. Visualisations should be designed in a way that fully encapsulates the state that can be returned to and re-rendered, as well as applied to subsequent steps. Tools should assume that users want to go back and forth both within and across visualisations, and might want to return to a previous state. Accordingly, efforts on visualisation provenance can suggest techniques for preserving and navigating states of an analysis, elevating the importance of analytic *processes* to have equal or greater importance to analytic *results* [60].

A stateful approach to visualising traces would allow for a history of interactions and for users to be able to effectively undo previous transformations on the data. Implementing this robustly will also aid in inter-linking (**G4**) visualisations, which can be achieved by sharing the state of the previous visualisation to use as a starting point for the next.

History for reproducibility. A history of interactions with tools and the associated state would also allow analysts to share their analyses in a much more in-depth manner, showing another user exactly how they arrived at a certain result. This would also be beneficial when trying to introduce new users to the tool.

History as feedback for visualisation developers. If interaction history is recorded, it can be leveraged by visualisation developers to gain insights about user interactions. visualisation developers could leverage usage data to understand how users interact with systems, without having to rely on time-consuming qualitative interviews. Moreover, as described in §4, current DT users are disparate, which reduces the amount of communication and visibility between developers and users. Automating this feedback is therefore a compelling solution.

For example, if tools offer multiple views of the same data and customisation options, the time spent in each view and the selected options should be recorded. This data can be leveraged for understanding common user settings and correlating different user groups and levels of experience. It is also an opportunity to better understand the complex interactions a user has *between* various tools, by recording entry and exit points or, if possible, which tools are accessed immediately before and after the current tool.

This guideline can be informed by related work on monitoring users' interactions [61] and work on building in feedback [62]. This data could also be directly applied back to the visualisation to show users popular interface options, along with other augmentations, such as those presented in the work on Scented Widgets [63].

Recording interaction history can empower developers to address **C1** by understanding what users actually use and find useful. It can also indicate prevalence of certain actions and inform developers of the varying use cases as we describe in **C4**. It would also address **C2** and help users to communicate analyses to other people more clearly and with less effort. There is potential for work in the area of explanatory user interfaces [64] to lower the barrier to entry for new users.

7.6 G6: Aggregate Context (C2, C3)

Participants repeatedly highlighted the importance of aggregate context for extracting the value from DT visualisations, noting the necessity of aggregate data for deciding which traces were important to inspect. To reduce reliance on mental models we propose that, where possible, developers should integrate aggregate data into single trace views to provide as much context as is feasible about the trace. A concrete example would be to display per span latency distributions within a single trace visualisation. This reduces the mental burden on expert users, and novice users can begin to have some of the same insights as experienced users without relying on years of experience to build up a deep contextual mental model. This also makes things faster, as users often go back and forth between aggregate and non-aggregate data.

This guideline addresses **C2** and **C3** by objectively connecting data in aggregate to the single trace view and not relying on a user to infer this information from their mental model. Concepts such as external cognition [65] and focus+context [29] coupled with work on embedding visualisations in-situ could aid in reducing the cognitive load on users.

Aggregate context can be drawn from a variety of different populations and can be stratified along many different dimensions. For example, it could consider all traces ever, only traces containing certain services, or traces from the last 24 hours, to name some options. Prior work has described a need to stratify by diverse features such as geographic region, operating system version, and machine type [9]. Future work must solve the problem of whether to prevent the user from being overwhelmed with multiple different visualisations, facilitate customisation of the aggregated population, or visualise the uncertainty of the corpus that is being used.

7.7 G7: Comparison (C2, C3)

We observed that in practice, comparison is commonplace, even for tools with no explicit support; however, it is encumbered by the need to manually configure multiple visualisations to display the same information side-by-side. When comparison is treated as a first class use case and combined with **G1** and **G4**, this will greatly increase the efficiency at which users can carry out their tasks as they are no longer forced to work around the limitations of working in multiple tabs and relying on visual memory rather than being able to directly compare [66].

This guideline is intended to address **C2** and **C3** by encouraging visualisation developers to directly integrate common tasks such as comparison into their tools. Beyond making it easy for users to externally compare two instances of a visualisation, there is also an opportunity for special-purpose tools for trace comparison. Some work has explored this topic [16], [67] , but most of the proposed solutions are unwieldy and are not widely used or available in the tools that we observed.

7.8 G8: Handle Edge-Cases Gracefully (C2, C5)

The primary use case for DT is troubleshooting. Troubleshooting inherently requires investigating executions that are outside the norm, where something may have gone wrong during execution up to and including capturing the data itself. Ironically, trace visualisations often focus on 'golden case' data and are ill-suited to investigating edge-cases. Two edge-cases in particular are insufficiently supported yet frequently encountered by users of existing solutions: large traces and malformed data.

G2 proposes an approach for minimally representing an individual trace which we believe also has the potential to help with displaying large traces, opening up opportunities to redesign the visualisation and UI to replace the large areas of redundant information – such as those experienced by traces exhibiting fanout (repeated calls of the same service), a common culprit in large traces – with a simpler visualisation, and only expanding areas of the trace that a user actually wants to interact with. Future work in visualising large traces could also benefit from utilising large graph presentation and processing algorithms [68].

Addressing malformed data requires more explicit support from the trace collection system, to ensure that it flags when data is dropped or corrupted. Then, UIs can visually indicate the occurrence, rather than relying on the user to infer that the data is malformed. Prior work in visualisation can inform these efforts. A recent study evaluated alternative approaches to imputing and visualising missing data, finding that explicitly highlighting missing values led to better perceived data quality than breaking visual continuity [69]. Hence, work investigating visualising missing data could guide future work in this area.

A more general principle here is that investigating large traces and malformed data is a *crucial* part of analysis carried out by users and so, much like comparison, it should also be treated as an important aspect in the development of the tool. In doing so developers can address **C2** by not relying on users to simply know when data is malformed, and can begin to address **C5** with more complex approaches to interacting with malformed data.

8 Analysis of Existing Tools

To understand how well the current state-of-the-art tooling satisfies the needs of DT users we derived a small survey to illustrate the capabilities of existing widely-used systems. The questions were derived from our open discussions and helped inform the guidelines in §7. Table1 summarises the results and shows which capabilities drove specific guidelines and how well existing solutions satisfy these guidelines. The survey comprises some of the most widely used DT tools available (4 open source, 8 commercial) and the tools used by our experts (Comp.A,B). For commercial tools we gleaned answers in two main ways: some tools such as Honeycomb and Lightstep offer interactive sandboxes; for others we relied on documentation and video presentations to ascertain functionality. In some cases, marked as 'unclear', we were unable to ascertain the answer due to a lack of information.

Overall we observe that while some tools satisfy some guidelines, no tool satisfies all; moreover, the tools used by our expert users are neither significantly better nor worse than the existing open-source or commercial tools.

All of the tools offer very limited customisation and persistence (**G1**). Additionally, tools consistently failed to address presentation of large traces (**G8**). Despite the importance to users, it was difficult to ascertain how well commercial solutions would deal with large traces or malformed traces; in sandbox environments the largest trace was only approximately 40 spans, compared to thousands of spans described by practitioners in our study.

Although users frequently raised the importance of malformed or corrupted traces, none of the tools provided information regarding how or if they could display them, with the exception of Sentry, which provided detailed focus on highlighting malformed traces and drawing users' attentions to areas of traces where data was suspected to be missing.

The ability to search within a trace (**G3**) is, with the exception of DynaTrace, exclusively limited to searching on service name if at all, and does not leverage any other data from a span.

One area where commercial tools performed better than our experts' tools was switching between tools (**G4**). This occurred because commercial tools package 'comprehensive' all-in-one solutions to facilitate trace discovery and inspection in a single tool. As outlined in §7.4 this does improve user experience, but is a brittle approach as it ties users into performing all their analysis in one tool or suite of tools – something we explicitly saw our users did not want. None of the tools facilitate programmatic access to the underlying data in any straightforward manner.

Commercial solutions also better support aggregate comparison (**G7**) and identifying outlier traces (**G6**) and even spans (New Relic, Lightstep). However, when examining an individual trace this aggregate context is then lost and cannot be seen. The only exception is Honeycomb which provides comparison to the aggregate within the individual trace view. Removing this aggregate context makes it hard for users, especially new ones, to understand whether a trace or span is performing as expected.

Finally, no tool supports comparing 2 individual traces or 2 sets of traces, despite being a key use case identified by our users.

9 Discussion

9.1 Limitations

The major limitation of our work is the small number of participants. As detailed in §3.1, when seeking experienced users this is an unavoidable byproduct of the nascency of DT as a domain area. Additionally, having recruited from two major internet companies there is a caveat that our results are most relevant to these large, real-time environments. We have mitigated the risk of biasing our results towards these specific companies in two ways.

Firstly, the companies we recruited from are at the forefront of practically using DT and represent the state-of-the-art in utilising open-source (company B) and bespoke (company A) solutions. We have not recruited from companies using obscure solutions.

Secondly, the specific participants we recruited are experts in the DT domain area. All participants have extensive experience, working with more than one solution and though our results may still be skewed towards specific solutions our participants *currently* use, we encouraged them to speak in as general terms as possible to ensure more broadly relevant results. Additionally, **P1** and **P5** have both been actively involved in the implementation and maintenance of two of the most widely used open-source DT solutions and have deep knowledge of the current state-of-the-art as a result.

Finally, in §4 we establish two types of users, before going on to mainly focus on consumers in §5. Although only two of our participants represent pure consumers, potentially skewing results to their experiences, all participants have long-term experience working as consumers with the majority still actively doing so (and not just developing or maintaining them).

9.2 Open Problems

Through this work we have exposed several challenges and corresponding future research directions. In this section we extract several research topics that we believe can have a broader impact beyond just the DT application domain.

Multi-Variate Comparison. Trace comparison is analogous to work on graph comparison [70]. Trace comparison adds significant complexity as the data is multivariate, with multiple possible dimensions to stratify for comparison. Rather than just focusing on structural differences between generated graphs, further problems are exposed such as temporal differences and parent-child relationships. Approaches in dynamic graph presentation [71] and temporal graph visualisation [72] may guide approaches, but it remains an open research direction.

Interactions between different tools. Analysts' workflows involve an ecosystem of different tools rather than a single tool. This is prevalent in many other fields that touch on data analytics or data-driven decision making. Research has addressed this problem by facilitating multiple linked-view visualisations [73], dashboard design [74], as well as more contemporary work bridging the gap between data and written documents to ease workflows [75]. Within DT, companies have often developed fully integrated, single tools. However, none of these effectively solve the recurring problem of users wanting to interact with multiple standalone tools that leverage the same data and facilitate stateful interaction, consistent between investigations. Recent work has examined this problem when working with repeating text across multiple documents and tools [76] as well as beginning to apply these concepts to visualisation [77]. It still remains a rich and unsolved problem area. Standardisation may offer a solution, and the growing use of the OpenTelemetry [4] DT standard is an encouraging sign of progress towards more homogeneous and structured data.

Customisation of visualisation. It is difficult for a visualisation to solve all problems for all users. Customisation has attracted focus in dashboard presentation [78], but allowing a user to alter the visual representation of data has received less attention outside of classical approaches like bar and scatter plots [79]. There is often an inherent tradeoff between customisation and usability. For instance, this is exemplified by recent efforts in visualisation development tools such as d3 and vega-lite [80], [81], where the level of customisability tends to be inversely proportional to the level of coding effort required. There is opportunity to more deeply explore and manage this space of tradeoffs in distributed tracing visualisation and other application areas. This further introduces questions surrounding persistence and coordination – should customisation be persisted based on the user's profile, or on the data that they are interacting with?

A sub problem of customisation is how to hide information whilst giving an honest representation. This has implications in fields such as visualising uncertainty in data [82], and diverse customisation for differing personas [83].

Large data interactions. No tools adequately supported interacting with large traces. There is a physical limit to what can be displayed on the screen and the data itself is so large that processing time may become prohibitive. Novel techniques for navigating within large traces may find impact in other application domains, e.g. text visualisation – looking for specific passages, searching legal documents, analysing language used in reporting. Conversely, prior approaches may apply to DT, such as searching and filtering [84], explorative lenses [85], and decluttering techniques such as temporal distortion [86]; though there remain open questions, e.g. should lenses be a linked view or directly integrated. Similarly, prior work on progressive visualisation [87] may likewise inform scalability of large trace visualisation and suggest smarter pre-fetching solutions.

Visualisation is useful for helping users find patterns in rich, structured data; formalising this is a compelling future direction. Other application domains deal with multi-faceted and large-scale data, and inspiration may be found in areas such as multivariate graph searching [88] and multivariate matrix navigation [84].

User interaction history. User interaction history is often stored (although seldom in DT). How can this be leveraged? Are there ways to take the same repeated actions and re-use them on new data and reduce analysts' wasted effort? Furthermore, effectively linking data back to developers to inform their future design decisions is another open area. Work in this area could similarly feed usage data back to end-users to display how other users of the tool are using it [63], thus helping novice users too.

Although we focused our findings on the context of visualisation, they also offer implications and opportunities for future directions in areas such as algorithmic support in trace comparison and processing of large data before and during visualisation.

10 Conclusion

In this paper we presented the first characterisation of distributed tracing tools, derived from an interview study with six expert practitioners from two large internet companies. We presented coded lists of user groups, use cases, and challenges of existing tools; from these we derived eight guidelines for designing future distributed tracing tools. We surveyed state-of-the-art open-source and commercial distributed tracing tools and found that the majority of guidelines remain unsatisfied. Lastly, we highlighted five open problem areas that are relevant to distributed tracing and are also compelling future directions for visualisation research in general. Ultimately we hope our work opens up the application domain of distributed tracing both to more principled design and novel future visualisation contributions.

References

[1] A. Parker, D. Spoonhower, J. Mace, B. Sigelman, and R. Isaacs, *Distributed tracing in practice: Instrumenting, analyzing, and debugging microservices*. O'Reilly Media, 2020.
[2] Uber, "Jaeger," Retrieved Aug. 2021 from https://www.jaegertracing.io.
[3] Apache, "Zipkin," Retrieved Aug. 2021 from https://zipkin.io.
[4] OpenTelemetry, "OpenTelemetry," Retrieved Aug. 2022 from https://opentelemetry.io.
[5] W. W. W. C. (W3C), "Trace Context," Retrieved Aug. 2021 from https://www.w3.org/TR/trace-context/.
[6] "Lightstep," Retrieved Sept. 2021 from https://lightstep.com.
[7] "Honeycomb," Retrieved Aug. 2022 from https://www.honeycomb.io.
[8] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson *et al.*, "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems," in *Proc. Twenty-Fourth Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 3–18.
[9] J. Kaldor, J. Mace, M. Bejda, E. Gao, W. Kuropatwa, J. O'Neill, K. W. Ong, B. Schaller, P. Shan, B. Viscomi *et al.*, "Canopy: An end-to-end performance tracing and analysis system," in *Proc. 26th Symp. on Operating Systems Principles*, 2017, pp. 34–50.
[10] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag, "Dapper, a large-scale distributed systems tracing infrastructure," 2010.
[11] R. Fonseca, G. Porter, R. H. Katz, and S. Shenker, "X-trace: A pervasive network tracing framework," in *4th USENIX Symp. Networked Systems Design & Implementation (NSDI 07)*, 2007.
[12] I. Beschastnikh, P. Liu, A. Xing, P. Wang, Y. Brun, and M. D. Ernst, "Visualizing distributed system executions," *ACM Transactions Software Engineering and Methodology (TOSEM)*, vol. 29, no. 2, pp. 1–38, 2020.
[13] R. R. Sambasivan, I. Shafer, J. Mace, B. H. Sigelman, R. Fonseca, and G. R. Ganger, "Principled workflow-centric tracing of distributed systems," in *Proc. 7th ACM Symp. Cloud Computing*, 2016, pp. 401–414.
[14] C. Sridharan, "Distributed Tracing – we've been doing it wrong," Retrieved Sept. 2021 from https://copyconstruct.medium.com/distributed-tracing-weve-been-doing-it-wrong-39fc92a857df, 2019.
[15] J. Users, "Jaeger UI Open Issues Page," Retrieved Aug. 2022 from https://github.com/jaegertracing/jaeger-ui/issues/, 2022.
[16] J. Farro, "Trace comparisons arrive in Jaeger 1.7," Retrieved Aug. 2021 from https://medium.com/jaegertracing/trace-comparisons-arrive-in-jaeger-1-7-a97ad5e2d05d, 2018.
[17] J. Trümper, J. Bohnet, and J. Döllner, "Understanding complex multi-threaded software systems by using trace visualization," in *Proc. 5th int. symp. on Software vis.*, 2010, pp. 133–142.
[18] W. De Pauw and S. Heisig, "Zinsight: A visual and analytic environment for exploring large event traces," in *Proc. 5th int. symp. on Software vis.*, 2010, pp. 143–152.
[19] F.-G. Ottogalli, C. Labbé, V. Olive, B. de Oliveira Stein, J. C. de Kergommeaux, and J.-M. Vincent, "Visualization of distributed applications for performance debugging," in *Int. conf. computational science*. Springer, 2001, pp. 831–840.
[20] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.
[21] K. E. Isaacs, A. Giménez, I. Jusufi, T. Gamblin, A. Bhatele, M. Schulz, B. Hamann, and P.-T. Bremer, "State of the art of performance visualization." *EuroVis (STARs)*, 2014.
[22] K. E. Isaacs, P.-T. Bremer, I. Jusufi, T. Gamblin, A. Bhatele, M. Schulz, and B. Hamann, "Combing the communication hairball: Visualizing parallel execution traces using logical time," *IEEE Trans. Vis. Comput. Graph.*, vol. 20, no. 12, pp. 2349–2358, 2014.
[23] E. W. Bethel, H. Childs, and C. Hansen, *High performance visualization: Enabling extreme-scale scientific insight*. CRC Press, 2012.
[24] C. Xie, W. Xu, and K. Mueller, "A visual analytics framework for the detection of anomalous call stack trees in high performance computing applications," *IEEE Trans. Vis. Comput. Graph.*, vol. 25, no. 1, pp. 215–224, 2018.
[25] B. Gregg, "The flame graph," *Comm. ACM*, vol. 59, pp. 48–57, 2016.
[26] K. Xu, Y. Wang, L. Yang, Y. Wang, B. Qiao, S. Qin, Y. Xu, H. Zhang, and H. Qu, "Clouddet: Interactive visual analysis of anomalous performances in cloud computing systems," *IEEE Trans. Vis. Comput. Graph.*, vol. 26, no. 1, pp. 1107–1117, 2019.
[27] W. S. Cleveland and R. McGill, "Graphical perception: Theory, experimentation, and application to the development of graphical methods," *JASA*, vol. 79, no. 387, pp. 531–554, 1984.
[28] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *The craft of information visualization*. Elsevier, 2003, pp. 364–371.
[29] G. W. Furnas, "Generalized fisheye views," *Acm Sigchi Bulletin*, vol. 17, no. 4, pp. 16–23, 1986.
[30] C. North, "Toward measuring visualization insight," *IEEE computer graphics and applications*, vol. 26, no. 3, pp. 6–9, 2006.
[31] E. Wall, M. Agnihotri, L. Matzen, K. Divis, M. Haass, A. Endert, and J. Stasko, "A heuristic approach to value-driven evaluation of visualizations," *IEEE Trans. Vis. Comput. Graph.*, vol. 25, no. 1, pp. 491–500, 2018.
[32] J. Hoffswell, A. Satyanarayan, and J. Heer, "Augmenting code with in situ visualizations to aid program understanding," in *CHI Conf. Proc.*, 2018, pp. 1–12.
[33] D. Ceneda, T. Gschwandtner, T. May, S. Miksch, H.-J. Schulz, M. Streit, and C. Tominski, "Characterizing guidance in visual analytics," *IEEE Trans. Vis. Comput. Graph.*, vol. 23, no. 1, pp. 111–120, 2016.
[34] J. Walny, C. Frisson, M. West, D. Kosminsky, S. Knudsen, S. Carpendale, and W. Willett, "Data changes everything: Challenges and opportunities in data visualization design handoff," *IEEE Trans. Vis. Comput. Graph.*, vol. 26, no. 1, pp. 12–22, 2019.
[35] M. Engebretsen, H. Kennedy, and W. Weber, "Visualization practices in scandinavian newsrooms: A qualitative study," in *21st int. Conf. information Visualisation (iV)*. IEEE, 2017, pp. 296–300.
[36] L. E. G. Martins and T. Gorschek, "Requirements engineering for safety-critical systems: An interview study with industry practitioners," *IEEE Transactions Software Engineering*, vol. 46, no. 4, pp. 346–361, 2018.

[37] Y.-a. Kang and J. Stasko, "Characterizing the intelligence analysis process: Informing visual analytics design through a longitudinal field study," in *IEEE conf. visual analytics science and technology (VAST)*. IEEE, 2011, pp. 21–30.

[38] S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer, "Enterprise data analysis and visualization: An interview study," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 12, pp. 2917–2926, 2012.

[39] Z. Liu, B. Jiang, and J. Heer, "immens: Real-time visual querying of big data," in *Computer Graphics Forum*, vol. 32, no. 3pt4. Wiley Online Library, 2013, pp. 421–430.

[40] A. Rule, A. Tabard, and J. D. Hollan, "Exploration and explanation in computational notebooks," in *CHI Conf. Proc.*, 2018, pp. 1–12.

[41] A. Nurwidyantoro, M. Shahin, M. Chaudron, W. Hussain, H. Perera, R. A. Shams, and J. Whittle, "Towards a human values dashboard for software development: An exploratory study," in *Proc. 15th ACM/IEEE Int. Symp. Empirical Software Engineering and Measurement*, 2021, pp. 1–12.

[42] C. Zuo, L. Ding, and L. Meng, "A feasibility study of map-based dashboard for spatiotemporal knowledge acquisition and analysis," *ISPRS Int. Journal Geo-Information*, vol. 9, no. 11, p. 636, 2020.

[43] Z. Pan, C. Li, and M. Liu, "Learning analytics dashboard for problem-based learning," in *Proc. 7th ACM Conf. Learning@ Scale*, 2020, pp. 393–396.

[44] K. DeMarrais, "Qualitative interview studies: Learning through experience," *Foundations for research: Methods of inquiry in education and the social sciences*, vol. 1, no. 1, pp. 51–68, 2004.

[45] M. Sedlmair, M. Meyer, and T. Munzner, "Design study methodology: Reflections from the trenches and the stacks," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 12, pp. 2431–2440, 2012.

[46] J. C. Johnson and S. C. Weller, "Elicitation techniques for interviewing," *Handbook of interview research: Context and method*, pp. 491–514, 2002.

[47] J. S. Olson and W. A. Kellogg, *Ways of Knowing in HCI*. Springer, 2014, vol. 2.

[48] B. Chametzky *et al.*, "Coding in classic grounded theory: I've done an interview; now what?" *Sociology Mind*, vol. 6, no. 04, p. 163, 2016.

[49] J. W. Creswell and C. N. Poth, *Qualitative inquiry and research design: Choosing among five approaches*. Sage publications, 2016.

[50] N. Elmqvist and J. S. Yi, "Patterns for visualization evaluation," *Information Visualization*, vol. 14, no. 3, pp. 250–269, 2015.

[51] Q. Wang, P. Cavanagh, and M. Green, "Familiarity and pop-out in visual search," *Perception & psychophysics*, vol. 56, no. 5, pp. 495–500, 1994.

[52] T. M. Mann and H. Reiterer, "Evaluation of different visualizations of web search results," in *Proc. 11th Int. Workshop Database and Expert Systems Applications*. IEEE, 2000, pp. 586–590.

[53] F. Van Ham and A. Perer, ""search, show context, expand on demand": Supporting large graph exploration with degree-of-interest," *IEEE Trans. Vis. Comput. Graph.*, vol. 15, no. 6, pp. 953–960, 2009.

[54] B. Craft and P. Cairns, "Beyond guidelines: what can we learn from the visual information seeking mantra?" in *9th Int. Conf. Information Visualisation*. IEEE, 2005, pp. 110–118.

[55] G.-J. Ding, T. Hwang, and P.-C. Kuo, "Progressive disclosure options for improving choice overload on home screen," in *Int. Conf. Applied Human Factors and Ergonomics*. Springer, 2020, pp. 173–180.

[56] E. Horvitz, "Principles of mixed-initiative user interfaces," in *CHI Conf. Proc.*, 1999, pp. 159–166.

[57] E. L. Hutchins, J. D. Hollan, and D. A. Norman, "Direct manipulation interfaces," *Human–computer interaction*, vol. 1, no. 4, pp. 311–338, 1985.

[58] C. Ahlberg, C. Williamson, and B. Shneiderman, "Dynamic queries for information exploration: An implementation and evaluation," in *CHI Conf. Proc.*, 1992, pp. 619–626.

[59] T. K. Attwood, D. B. Kell, P. McDermott, J. Marsh, S. Pettifer, and D. Thorne, "Utopia documents: linking scholarly literature with research data," *Bioinformatics*, vol. 26, no. 18, pp. i568–i574, 2010.

[60] C. North, R. Chang, A. Endert, W. Dou, R. May, B. Pike, and G. Fink, "Analytic provenance: process+ interaction+ insight," in *CHI'11 Extended Abstracts on Human Factors in Computing Systems*, 2011, pp. 33–36.

[61] J. Alexander, A. Cockburn, and R. Lobb, "Appmonitor: A tool for recording user actions in unmodified windows applications," *Behavior Research Methods*, vol. 40, no. 2, pp. 413–421, 2008.

[62] S. McKenna, D. Staheli, and M. Meyer, "Unlocking user-centered design methods for building cyber security visualizations," in *2015 IEEE Symp. on Visualization for Cyber Security*. IEEE, 2015, pp. 1–8.

[63] W. Willett, J. Heer, and M. Agrawala, "Scented widgets: Improving navigation cues with embedded visualizations," *IEEE Trans. Vis. Comput. Graph.*, vol. 13, no. 6, pp. 1129–1136, 2007.

[64] A. García Frey, G. Calvary, and S. Dupuy-Chessa, "Xplain: an editor for building self-explanatory user interfaces by model-driven engineering," in *Proc. 2nd ACM SIGCHI symp. Engineering interactive computing systems*, 2010, pp. 41–46.

[65] M. Scaife and Y. Rogers, "External cognition: how do graphical representations work?" *Int. journal human-computer studies*, vol. 45, no. 2, pp. 185–213, 1996.

[66] T. Munzner, *Visualization analysis and design*. CRC press, 2014.

[67] R. R. Sambasivan, I. Shafer, M. L. Mazurek, and G. R. Ganger, "Visualizing request-flow comparison to aid performance diagnosis in distributed systems," *IEEE Trans. Vis. Comput. Graph.*, vol. 19, no. 12, pp. 2466–2475, 2013.

[68] M. Molina-Solana, D. Birch, and Y.-k. Guo, "Improving data exploration in graphs with fuzzy logic and large-scale visualisation," *Applied Soft Computing*, vol. 53, pp. 227–235, 2017.

[69] H. Song and D. A. Szafir, "Where's my data? evaluating visualizations with missing data," *IEEE Trans. Vis. Comput. Graph.*, vol. 25, no. 1, pp. 914–924, 2018.

[70] K. Andrews, M. Wohlfahrt, and G. Wurzinger, "Visual graph comparison," in *13th Int. Conf. Information Visualisation*. IEEE, 2009, pp. 62–67.

[71] F. Beck, M. Burch, and S. Diehl, "Towards an aesthetic dimensions framework for dynamic graph visualisations," in *13th int. conf. information visualisation*. IEEE, 2009, pp. 592–597.

[72] N. Kerracher, J. Kennedy, and K. Chalmers, "The design space of temporal graph visualisation." in *EuroVis (Short Papers)*, 2014.

[73] J. C. Roberts, "Exploratory visualization with multiple linked views," in *Exploring geovisualization*. Elsevier, 2005, pp. 159–180.

[74] A. Janes, A. Sillitti, and G. Succi, "Effective dashboard design," *Cutter IT Journal*, vol. 26, no. 1, pp. 17–24, 2013.

[75] Z. Chen and H. Xia, "Crossdata: Leveraging text-data connections for authoring data documents," in *CHI Conf. Human Factors Comput. Sys.*, 2022, pp. 1–15.

[76] H. L. Han, J. Yu, R. Bournet, A. Ciorascu, W. E. Mackay, and M. Beaudouin-Lafon, "Passages: Interacting with text across documents," in *CHI Conf. Human Factors Comput. Sys.*, 2022, pp. 1–17.

[77] L. Nonnemann, M. Hogräfer, H. Schumann, B. Urban, and H.-J. Schulz, "Customizable coordination of independent visual analytics tools," 2021.

[78] M. Elias and A. Bezerianos, "Exploration views: understanding dashboard creation and customization for visualization novices," in *IFIP conference on human-computer interaction*. Springer, 2011, pp. 274–291.

[79] B. Saket, H. Kim, E. T. Brown, and A. Endert, "Visualization by demonstration: An interaction paradigm for visual data exploration," *IEEE Trans. Vis. Comput. Graph.*, vol. 23, no. 1, pp. 331–340, 2016.

[80] M. Bostock, V. Ogievetsky, and J. Heer, "D^3 data-driven documents," *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 12, pp. 2301–2309, 2011.

[81] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer, "Vega-lite: A grammar of interactive graphics," *IEEE Trans. Vis. Comput. Graph.*, vol. 23, no. 1, pp. 341–350, 2016.

[82] J. Hullman, X. Qiao, M. Correll, A. Kale, and M. Kay, "In pursuit of error: A survey of uncertainty visualization evaluation," *IEEE Trans. Vis. Comput. Graph.*, vol. 25, no. 1, pp. 903–913, 2018.

[83] B. J. Jansen, J. O. Salminen, and S.-G. Jung, "Data-driven personas for enhanced user understanding: Combining empathy with rationality for better insights to analytics," *Data and Information Management*, vol. 4, no. 1, pp. 1–17, 2020.

[84] Y. Yang, W. Xia, F. Lekschas, C. Nobre, R. Krüger, and H. Pfister, "The pattern is in the details: An evaluation of interaction techniques for locating, searching, and contextualizing details in multivariate matrix visualizations," in *CHI Conf. Human Fact. Comput. Sys.*, 2022, pp. 1–15.

[85] R. Rao and S. K. Card, "The table lens: merging graphical and symbolic representations in an interactive focus+ context visualization for tabular information," in *CHI Conf. Proc.*, 1994, pp. 318–322.

[86] B. Morrow, T. Manz, A. E. Chung, N. Gehlenborg, and D. Gotz, "Periphery plots for contextualizing heterogeneous time-based charts," in *IEEE visualization conf. (VIS)*. IEEE, 2019, pp. 1–5.

[87] J.-D. Fekete, D. Fisher, A. Nandi, and M. Sedlmair, "Progressive data analysis and visualization," 2019.

[88] H. Kobayashi, H. Suzuki, and K. Misue, "A visualization technique to support searching and comparing features of multivariate datasets," in *19th Int. Conf. on Information Visualisation*. IEEE, 2015, pp. 310–315.

Thomas Davidson is a PhD candidate at the Max Planck Institute for Software Systems as part of the Cloud Software Systems Group.

Emily Wall is an assistant professor in Computer Science at Emory University, where she directs the Cognition and Visualization Lab.

Jonathan Mace is tenure-track faculty at the Max Planck Institute for Software Systems and head of the Cloud Software Systems Group.