Behavior-Driven Optimization Techniques for Scalable Data Exploration

by

Leilani Battle

B.S., University of Washington, Seattle (2011)
M.S., Massachusetts Institute of Technology (2013)

Submitted to the Department of
Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2017

Author . . . . . .
Department of
Electrical Engineering and Computer Science
May 19, 2017

Certified by . . . . . .
Michael Stonebraker
Adjunct Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Behavior-Driven Optimization Techniques for Scalable Data Exploration

by

Leilani Battle

Submitted to the Department of
Electrical Engineering and Computer Science
on May 19, 2017, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Interactive visualizations are a popular medium used by scientists to explore, analyze and generally make sense of their data. However, with the overwhelming amounts of data that scientists collect from various instruments (*e.g.,* telescopes, satellites, gene sequencers and field sensors), they need ways of efficiently transforming their data into interactive visualizations. Though a variety of visualization tools exist to help people make sense of their data, these tools often rely on database management systems (or DBMSs) for data processing and storage; and unfortunately, DBMSs fail to process the data fast enough to support a fluid, interactive visualization experience.

This thesis blends optimization techniques from databases and methodology from HCI and visualization in order to support interactive and iterative exploration of large datasets. Our main goal is to reduce latency in visualization systems, *i.e.,* the time these systems spend responding to a user's actions. We demonstrate through a comprehensive user study that latency has a clear (negative) effect on users' high-level analysis strategies, which becomes more pronounced as the latency is increased. Furthermore, we find that users are more susceptible to the effects of system latency when they have existing domain knowledge, a common scenario for data scientists. We then developed a visual exploration system called Sculpin that utilizes a suite of optimizations to reduce system latency. Sculpin learns user exploration patterns automatically, and exploits these patterns to pre-fetch data ahead of users as they explore. We then combine data-prefetching with incremental data processing (*i.e.,* incremental materialization) and visualization-focused caching optimizations to further boost performance. With all three of these techniques (pre-fetching, caching, and pre-computation), Sculpin is able to: create visualizations 380% faster and respond to user interactions 88% faster than existing visualization systems, while also using less than one third of the space required by other systems to store materialized query results.

Thesis Supervisor: Michael Stonebraker
Title: Adjunct Professor of Electrical Engineering and Computer Science

Acknowledgments

Of course, there have been many, many people who have supported me along this journey. I will do my best, but this is by no means the complete list.

First, I would like to thank my advisor, Mike Stonebraker, for constantly pushing me to do my very best as a researcher. I had several moments throughout my PhD where I was unsure whether I could do what was asked of me, but Mike reassured me and gave me the confidence I needed to keep going. I am grateful for the time and effort he has spent providing me with feedback and advice on anything and everything related to academia and research. Mike is amazing. I am also incredibly grateful for the space I have been given to try out my own ideas. Mike has always been patient in listening to my (often half-baked) ideas, and very helpful in turning some of these ideas into meaningful research projects. And thank you Mike for putting up with me running into your office at random times to give you an update, or to ask you (probably too many) questions. I look forward to moving on to the next stage of my research career with Mike as a mentor.

Second, I want to thank Remco Chang for his advice, support and unwavering enthusiasm for this work. Remco has been a wonderful collaborator and mentor to me since my masters degree, and I have learned so much about visual analytics, human perception, and human-computer interaction from him. Both Remco and Mike were always there for me, and I really enjoyed the mix of expertise that the three of us brought to our joint projects. I sincerely hope that we can continue to collaborate in the years to come.

Many thanks to Sam Madden for giving me great advice every single year I was in graduate school, from applying to the NSF GRFP my first year, to figuring out where to do a postdoc in my last year. Seriously, thank you. I would like to thank David Karger for being on my RQE committee, and for being on my thesis committee. David has given me incredible feedback and advice that has made my research stronger and richer. Thank you to Danyel Fisher for taking on a database student with only a little bit of visualization experience as a summer intern. I learned a ton, had a blast, and I hope we can collaborate again soon. Also, many thanks to Danyel for helping me brainstorm the idea that ultimately became the second chapter of this thesis. Many thanks to Magda Balazinska and Bill Howe

for introducing me to database research while I was an undergrad at UW, for encouraging me to attend graduate school, and for continuing to support me throughout my PhD at MIT.

James Frew was instrumental in designing and conducting the MODIS-based user studies in my thesis, for which I am very grateful. Thank you to Alex Poliakov and Paul Brown for helping me with my SciDB issues, and to Paradigm4 for supporting my work with SciDB in general. And thank you to the huge number of people who have participated in my user studies. My work could not have been done without them!

Thank you to everyone in the MIT Database group for being incredibly supportive. I will miss our Tuesday lunches and fun conversations. And in particular, thank you to Jennie Rogers, Eugene Wu, Adam Marcus, Aaron Elmore, Aditya Parameswaran, Alekh Jindal, Manasi Vartak, Rebecca Taft, Gary Planthaber, Neha Narula, Holger Pirk and so many others for helping me along the way. Thank you Sheila Marian for helping me with logistics. I also want to thank VALT for how welcoming and supportive they have been over the years, particularly Jared Chandler, Eli Brown, Alvitta Ottley and Jordan Crouser.

I want to thank some of my closest friends for supporting me throughout my PhD: Rachael Harding, Cindy Sung, Andrew Jones, Kendall Nowocin, Hillary Eichler, and Ariel Lenning. I am a stronger and more well-rounded person because of each and every one of you. Thank you to the MIT offices, organizations and student groups that I have relied on throughout my PhD, including: the Black Graduate Student Association, the Academy of Courageous Minority Engineers, the Office of the Dean for Graduate Education, the EECS Graduate Student Association, the Graduate Women of Course 6, the Graduate Women's Reading Group, My Sister's Keeper, the Graduate Student Council, the CSAIL Student Committee, the CSAIL Games Night group, Floorpi and the East Campus house team, the EECS Graduate Office, and Community Wellness.

My mother and father have been my anchors, my rocks. They have shown me what it truly means to be resilient. My younger sister and brother have always been there to laugh and cry with me. Thank you to my grandparents for helping to raise me and my siblings. I love my family with all my heart, and I cannot thank them enough.

Last but not least, thank you to my wonderful husband Ray for his love and support. We have been through so much together, and I am truly grateful to have him by my side.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

The physical and biological sciences are becoming more data driven, often due to overwhelming quantities of data collected from satellites, telescopes, sequencers, and other sensors. In many discussions with scientists across a variety of specialties, we have found that interactive visualizations are important tools for helping people make sense of massive amounts of data. In particular, interactive visualizations are critical in the early stages of data analysis, when a scientist is browsing a new dataset.

In this thesis, we focus on the specific case of *exploratory browsing*, where the user explores her data at multiple levels of granularity, or *zoom levels*, through a pan-zoom interface. We have found that scientists exhibit a specific search pattern during the exploratory browsing process. They start by exploring at coarse-grained zoom levels, looking for regions of interest to analyze. When they find something interesting, they zoom into this particular region, and analyze it in more detail. After thoroughly exploring a particular region, they then zoom out and repeat this process with new regions of interest.

One common approach to addressing the issue of scalability (*i.e.,* of visualizing larger datasets) is by designing or augmenting visualization tools so they can connect directly to a remote database management system (or DBMS). DBMSs are designed specifically to support efficient data processing at scale. In this way, the visualization tool can leave the task of processing the data to the DBMS by translating the user's analyses into DBMS

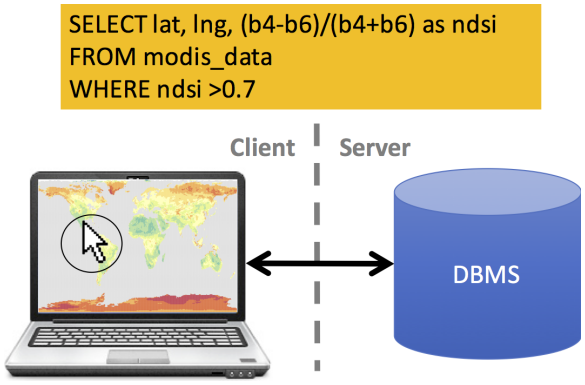


Figure 1-1: Typical architecture for an exploratory browsing system.

queries, and instead focus on efficiently rendering the query results from the DBMS. An example of the architecture for a basic exploratory browsing system is provided in Figure 1-1. The user interacts with a visualization tool running on a client machine (*i.e.,* the user's laptop, on the right side of Figure 1-1), and the client is connected to a DBMS running on a remote server (left side of Figure 1-1).

At the beginning of the exploratory browsing process, the user inputs a complex query that she wants to execute and then visualize. For example, the query at the top of Figure 1-1 is applying a snow cover calculation to raw satellite imagery data collected from the NASA Moderate Resolution Spectroradiometer (or MODIS). This calculation is called the Normalized Difference Snow Index, or NDSI, because it computes the normalized difference between two separate wavelengths of light: visible red light (represented as $b4$ in Figure 1-1), and near-infrared light (represented as $b6$ in Figure 1-1). The last line of the query filters the snow cover results, such that only high NDSI values are returned. The client-side visualization tool then sends the user's query to the DBMS to be prepared for visualization, which includes executing the query in the DBMS. We also refer to this data preparation step as *creating a visualization* of the user's query. After the user's query has been prepared for visualization, the user can then interact with a pan-zoom interface on the client to explore the rendered results. As the user interacts with the client-side interface, the visualization tool sends requests to the DBMS to retrieve the corresponding query results needed to update the current visualization.

Unfortunately, DBMSs fail to process the data fast enough to support a fluid, interactive visualization experience. This is because DBMS are not designed to provide query results

at interactive speeds [48, 10]. Specifically, many of the operations involved in complex analytics (*e.g.,* linear algebra operations) involve processing most if not all of the underlying input data. As such, complex analytical queries are nearly impossible for the DBMS to execute at interactive speeds when applied to massive datasets [96]. Since many visualization systems rely on DBMSs to scale up to larger datasets, their data processing speeds (and thus their overall performance) are severely limited as a result. Therefore a central theme of this thesis is to make visual exploration of massive datasets *interactive*, where we aim to have the system respond to user interactions (*e.g.,* pans and zooms) within 500ms.

1.2 Contributions

The main challenge in supporting interactivity lies in how visualization tools and DBMSs interact. Specifically, the DBMS is oblivious to useful contextual information collected by the visualization tool, and thus misses key opportunities for developing powerful performance optimizations to reduce system latency. By ignoring how people interact with visualizations, the DBMS loses valuable information about consistent interaction patterns and behaviors that can be exploited to adapt its optimization strategies. We show that by analyzing the design of the visualization interface and the user's interactions with this interface, along with applying various database optimizations that exploit this contextual information, we are able to achieve significantly better performance.

We developed a modular layer of software that acts as an intermediary between the visualization tool and the DBMS. Within this intermediate layer, we gain access to the user's interaction history, collected through existing visualization tools. We use this interaction data to develop models of user behavior, and then apply these models to develop context-aware (or visualization-aware) database optimization techniques within our intermediate layer. Within this software layer, we designed and implemented three key optimization techniques: 1) prediction techniques to anticipate the user's future interactions and prefetch the corresponding data ahead of time; 2) incremental data processing techniques to ensure that we only visualize the parts of the dataset that the user will ultimately explore; and 3) new caching techniques to maximize utilization of available cache space across both

the client and the server. In this way, we provide a re-usable software design that can be applied to a variety of visualization tools and database management systems, while still providing significant performance benefits.

When compared with existing systems, we can create visualizations from the user's query in a quarter of the time using incremental data processing; we can respond to user interactions (*e.g.,* pans and zooms) with this visualization in roughly half the time using smarter caching techniques and predictive pre-fetching; and we only use a quarter of the disk space required by other exploratory browsing systems to materialize the query results needed to produce visualizations (also through incremental data processing).

In the remainder of this section, we outline the major contributions of this thesis, which we divide into two parts: 1) we study how latency affects users' high level search strategies when exploring data using in pan-zoom interface (Chapter 2); and 2) we present our new visualization system Sculpin (Chapter 3), and discuss three major techniques implemented in Sculpin to reduce system latency (predictive pre-fetching (Chapter 4), visualization-aware caching (Chapter 5), and incremental data processing (Chapter 5)).

1.2.1 Measuring the Effects of System Latency

In order to effectively minimize the effects of system latency, we must first understand how latency actually influences the user as she visually explores her data. To do this, we conducted a comprehensive user study, where we had over 800 participants explore a collage of images using a tile-based visualization interface, similar to Google Maps. We explain our user study and our results in detail in Chapter 2. In this study, users completed search tasks under various latency and analysis conditions. Our results show that when latency is introduced in the interface, users change their search strategies and even ignore regions of the dataset that are too "costly" (or slow) to explore. Furthermore, we observed that latency had a noticeable effect on participants in only certain analysis contexts, the most significant being when the user already has domain knowledge that can help them better navigate the underlying dataset. This particular analysis context represents a well-known scenario in the real world, where data scientists are analyzing and exploring a dataset.

1.2.2 Reducing the Effects of System Latency

We then focus our attention on developing optimization techniques to reduce overall system latency, and thus reduce the negative impact of latency on exploration tasks. To do this, we developed Sculpin, a general-purpose tool for exploratory browsing of large datasets. Sculpin utilizes a client-server architecture, where the user interacts with a lightweight client-side interface to browse datasets, and the data to be browsed is retrieved from a DBMS running on a back-end server. We assume a detail-on-demand browsing paradigm, and optimize the back-end support for this paradigm by inserting a separate middleware layer in front of the DBMS. We discuss the Sculpin architecture in detail in Chapter 3. Within Sculpin's middleware layer, we have implemented three optimization techniques:

1. To make individual interactions (*i.e.,* pans and zooms) fast, the middleware layer predicts the user's future interactions and fetches the corresponding data ahead of the user as she explores a dataset. We consider two different mechanisms for prefetching: (a) learning what to fetch from the user's recent movements, and (b) using data characteristics (e.g., histograms) to find data similar to what the user has viewed in the past. We incorporate these mechanisms into a single prediction engine that adjusts its prediction strategies over time, based on changes in the user's behavior. We discuss our prediction techniques in Chapter 4.

2. With more information about the user's exploration behavior, we can also develop specialized caching strategies to further improve performance. For example, we can augment existing cache replacement policies (*e.g.,* a least-recently used policy) to take into account information about the user's recent interactions (*e.g.,* evict tiles corresponding to unlikely interaction sequences). We developed a suite of visualization-aware caching strategies to efficiently manage materialized query results from the DBMS. We discuss our caching techniques in Chapter 5.

3. Last, we aim to reduce the latency incurred when preparing the user's query for visualization, which is dominated by the time required to execute this query in the DBMS. We can simultaneously reduce both system latency and disk space consumption by applying incremental query execution strategies to avoid processing the parts

of the user's DBMS query that will never be explored. We discuss our incremental data processing techniques in Chapter 5.

Even though Sculpin includes three different optimization techniques to improve performance, our prediction techniques act as an anchor to support all of these optimizations. As such, we evaluated Sculpin through two separate user studies: 1) one study to assess the viability of prediction as a suitable optimization technique for visualization use cases (Chapter 4); and 2) a follow-up study to evaluate the performance of all of the optimization techniques used in Sculpin (Chapter 5). To ensure that our user studies were a close approximation real-world use cases, we recruited scientists directly for both studies, and all of our participants explored several real-world datasets as part of these studies.

In the first study, we found that our dynamic prefetching strategy provides: (1) significant improvements in overall latency when compared with non-prefetching systems (430% improvement); and (2) substantial improvements in both prediction accuracy (25% improvement) and latency (88% improvement) relative to existing prefetching techniques.

In the second study, we evaluated all three of our techniques. With our incremental pre-computation techniques, we found that Sculpin was able to provide significant improvements in both the materialization latency (380% improvement), and disk space used to store data tiles (370%). Furthermore, we found that our cache-optimization techniques provided an additional 200ms (or 60%) reduction in response times, when used in conjunction with our other techniques. When these optimizations are combined, Sculpin provides a 370% improvement over existing systems, while also supporting interactive exploration of multidimensional data (average response time of 490ms or less).

Chapter 2

The Effects of Latency on Visual Search Strategies for Different Analysis Tasks

2.1 Introduction

A central tenet of designing interactive visualization systems is to make the system responsive – that is, to reduce or remove latency in a system's response to the user's interactions. Because interactions in a visualization system have been thought to be an externalization of a user's analysis process [79], disruption to the process due to system latency can significantly impact the user's ability to maintain "cognitive flow" [24, 36, 84, 18], thereby reducing their analysis capabilities. In a recent paper by Liu and Heer, the authors find that latency beyond 500ms can make a visualization appear unresponsive to the point of being unusable by the users [62].

However, while there is a common belief that latency in visualization tools is disruptive, there have been conflicting reports of the effects of latency in other domains. For example, introducing latency can decrease the user's performance in interaction tasks [65, 101], reduce a user's sense of presence in a virtual environment [67], and affect a user's behavior with a computer system [62, 6]. On the other hand, Claypool found no effect of latency on a player's performance in real-time strategy (RTS) games [22, 89]. Similarly, Meehan et al. found that while latency has an effect on a user's search behavior on the web, the effect is only observed when the latency is above a certain threshold (*e.g.,* 1000ms) [6]. These

anecdotal but contradictory findings suggest that the effect of latency can depend on the user's task, the context, and the amount of latency.

While Liu and Heer have established that in using the *brushing and linking* technique in a visualization requires latency of less than 500 milliseconds [62], little is known about how latency can influence a user's higher-level analysis goals. In this chapter, we explore the effects of latency on visual search strategies for a range of analysis tasks. Complementary to prior work, which focuses on the effect of latency on low-level interactions, our goal is to examine how latency can affect a user's strategies in different analysis tasks and scenarios. Our hypothesis is that **the effects of latency on user behavior will be different depending on the analytic scenarios**. As such, a major contribution of this chapter is to identify the analysis scenarios where system latency has a clear effect on user behavior, and to define the nature of these behavioral shifts.

To validate our hypothesis, we first conducted a literature survey and compiled a list of 8 common analysis scenarios. From these analysis scenarios, we conducted an 8 (analysis scenario) x 5 (amount of latency) factorial design experiment to evaluate how latency affects a user's analysis strategy. Using a Google Maps-like interface, participants conducted visual search tasks where they were instructed to locate two specific "target images" in a collage of distracting images (see Figure 2-1). In this collage, the distractor (non-target) images can be slow to appear—that is, there is a controlled latency to rendering these images. In our experiment, we systematically control the amount of latency of these distractor images, and observe whether increasing the latency can cause a user to avoid the high latency (*i.e.,* slow rendering) regions of the collage while seeking the target images.

Our results show that, as expected, latency causes the users to avoid regions of the collage that are slow to render. Furthermore, we find that the effect of latency is not universal across all analysis scenarios. In some cases, latency can significantly alter the user's behavior; but in other cases, the effect is less noticeable. These findings support our original hypothesis. In addition, we analyzed the users' analysis search paths under the different analysis scenarios and latency conditions. We found five unique analysis strategies that were utilized by participants, as well as clear differences in which strategies were chosen for different analysis tasks and different latencies observed in the interface. These results

together indicate that latency in a system can significantly alter a user's search behavior. Furthermore, the user's experience level (novice vs. expert) and the nature of her expertise (domain knowledge vs. interface and task knowledge) can alter the degree to which system latency will have an effect. These results indicate that a specific class of users are likely to be more susceptible to the effects of system latency in visualization tools: domain experts, such as data analysts and data scientists. Therefore, new optimization techniques are needed for visualization tools, to mitigate or reduce the effects of system latency for this class of expert users.

In summary, this chapter makes three primary contributions:

1. Grounded in a literature review of various analytical domains, we provide a description of 8 high-level analysis scenarios.
2. We describe the results of a comprehensive experiment that demonstrates the effects latency across these scenarios.
3. Our results show that system latency has a clear effect on user's search behavior, and this effect becomes more pronounced as latency is increased. Furthermore, we show that users with domain knowledge are more likely to be affected by latency, a common scenario for expert users such as analysts and data scientists.

This chapter is outlined as follows. In Section 2.2, we discuss our literature review, which describes 8 different types of analysis tasks. In Section 2.3, we provide the details for our 8 x 5 factorial experiment, and we present experimental results per analysis task in Section 2.4. We present results from a global statistical analysis of the experimental results in Section 2.5, a global analysis of interaction trails in Section 2.6, and a discussion of our results in Section 2.7. We summarize the outcomes for this chapter in Section 2.8.

2.2 Literature Review: Analysis Scenarios

In order to evaluate the effects of latency on various analytical tasks, we first conducted a literature review to identify canonical analytical scenarios. We began with seminal work in intelligence analysis by Heuer [42] and from there we branched to relevant works across psychology, HCI, and VIS. Below we present a sample of 8 high-level analytical scenarios

along with real-world examples.

2.2.1 Novice analysis

As a baseline condition, we first consider a scenario in which an analyst has little knowledge of either the task or the domain. Early work in understanding cognitive models of analysis identified that novice analysts tend to exhibit an initial problem-scoping phase, followed by more detailed reasoning [95]. This work led to the hypothesis that poor performance by novice analysts could be ascribed to failure to scope the problem, poor formation of a conceptual model of the problem domain, or insufficient testing of hypotheses [95]. By developing a better understanding of novice problem-solving behavior under various constraints, we may be able to identify *specific error-prone behaviors* that may limit the novice's ability [87], and in turn help novice analysts to avoid these behaviors by providing better tools for conceptualizing and exploring the problem domain [12].

2.2.2 Analysis under time pressure

While a leisurely and substantial orientation phase may enable a novice analyst to develop a thorough understanding of the problem, in real analytical environments one rarely has the luxury of unlimited time. If pressured to make a decision quickly, it has been observed that people tend to lock in on a single strategy and demonstrate decreased ability to effectively evaluate alternative strategies [31]. Moreover, time constraints can cause people to revert to a *known strategy*, even if these same people employ a more logical strategy when the time constraint is removed [74]. We've learned that time pressure also has a predictable effect on decision time in a visual search task. Specifically, the degree of urgency appears to influence the threshold at which a detected signal triggers a response [82]. We incorporate time constraints as a separate analysis scenario in our study.

2.2.3 Expert analysis

Though there are many systems designed to support analysis by novices, it is frequently the case that the analyst brings with her some form of expertise. Making an important

distinction in the context of negotiation, Professor Margaret Neale of Stanford noted that "experience is feedback, expertise is strategic conceptualization" [70]. This strategic conceptualization may be further broken down into technical expertise and domain-specific background knowledge [68], and it has been observed that the influence of these two types of expertise on search tasks is not identical [43]. We incorporate one analysis scenario for the technical expertise case, and one scenario for the domain expertise case in our study.

Technical expertise

Technical expertise and competencies related to the task (rather than to a specific domain) are employed in many analytical contexts. For example, a canine unit trained to detect trace amounts of explosives is an effective partner in disrupting terrorist activities [34], despite the fact that the dog has no understanding of the complex sociopolitical implications of his task. In search tasks conducted by humans, facility with technical tools results in measurable differences in query complexity, target selection, post-query browsing, and the ultimate success rate relative to those with less advanced technical skills [105], regardless of the familiarity with the domain.

Domain expertise

Domain expertise stems from a deep, sometimes tacit understanding of a specific topic or discipline. It has been observed that domain experts search differently than people with limited domain knowledge [104]. Specifically, this kind of expertise has been observed to have a “honing” effect on search behavior: those with greater familiarity with a topic are more efficient in seeking out relevant information [49] and have higher rates of success in finding what they are looking for than non-experts. In studies of visual search in the context of chess masters, there is strong evidence that it is a *perceptual encoding advantage* stemming from their intimate knowledge of the domain, rather than a general perceptual or memory superiority, that enables their superior performance[83]. This perceptual advantage manifests through an increased sensitivity to semantic changes *in images within their domain of expertise* [103]. It does not, however, mitigate the effects of inattentional blindness [29].

2.2.4 Decision Making under Uncertainty

In many real-world analytical scenarios, particularly those involving streaming data, an analyst must move forward with their analysis despite uncertainty in their current knowledge of the domain. Of particular interest is the observation that decision-makers distinguish between three types of uncertainty [61]: inadequate understanding (i.e. making an incorrect assumption), incomplete information (i.e. having a rough idea but not exact knowledge), and undifferentiated alternatives (i.e. having a clear understanding of multiple paths, but lacking a mechanism for determining which leads to pursue). The first case (inadequate understanding) is outside of the control of our study, so we do not elaborate on it here. We have incorporated the other two cases (incomplete information and undifferentiated alternatives) as separate analysis scenarios within our study.

Analysis with incomplete information

Missing information is unavoidable in nearly all real-world analytic environments. In the context of evaluating consumer products, it has been shown that many people have a tendency to over-value categories of information that are present across all data points, intrinsically presuming the worst in missing values [52]. A similar effect has been observed in participants asked to evaluate applicants for a hypothetical scholarship [54]. When avoiding or ignoring missing information fails, individual differences in methods for coping with missing information as well as the assumptions people make in trying to fill in the gaps can have a significant impact on accuracy [30].

Analysis with undifferentiated alternatives

Having multiple valid, equally-plausible courses of action is perhaps one of the most ubiquitous and frustrating manifestations of uncertainty. Experts must painstakingly weigh the potential outcomes of regulatory and public policy decisions [11, 55], and laypersons may experience similar tension in more personal decisions such as which house to buy [66] or which course of cancer treatment to pursue [38]. In each of these cases, the absence of a clear winner impels the decision-maker to find a useful heuristic, or otherwise to simply

guess.

2.2.5 Opportunistic analysis

In some scenarios, the analyst is presented with a plurality of potential targets rather than only a few. For example, consider the use of mobile dating applications such as Tindr [45]. When many viable opportunities present themselves simultaneously, an analyst may opportunistically "hop" from target to target, resulting in identifying nearby, potentially related targets before branching out and exploring more distant regions of the dataset. We represent this case as a unique analysis scenario in our study.

2.2.6 Analysis at scale

Of particular interest in many current applications in data science are the analytical strategies employed in the context of big data [13]. When faced with a large amount of data, an analyst may not have the time to examine all the data but will need to focus on specific areas. As a result, their information seeking behavior can be substantially different. Therefore, we include analysis at scale as a separate analysis scenario in our study.

2.2.7 Summary

In this section, we identified 8 different analysis scenarios that people commonly encounter in the real world. These scenarios can be grouped by the user's expertise: analysis by novice users; analysis by users with technical (*i.e.,* task or interface) expertise; and analysis by users with domain (*i.e.,* data) expertise. In the novice case, we described three scenarios: a novice exploring a new dataset (*i.e.,* our base condition); a novice exploring under time pressure; and a novice exploring a large-scale dataset. We describe one scenario for technical expertise, where the user has knowledge of the search interface. We then discuss four cases of domain expertise: complete domain knowledge (*i.e.,* perfect information), incomplete domain knowledge, undifferentiated alternatives (*i.e.,* domain knowledge with uncertainty), and opportunistic knowledge (*i.e.,* knowledge gained directly through data exploration). Each scenario was selected based on how users' perceptions of the dataset

and task may differ from the novice case. In the next section, we describe our experimental design, and how we measure differences in user behavior for each of these different analysis scenarios.

2.3 Experiment

We know that latency can impact users' experiences with visual analytics tools. At the interaction level, users notice and even avoid specific interactions with high latency (e.g., brushing and linking with 500 milliseconds of latency introduced [62]). However, little work has been done to evaluate how latency may affect a user's decisions throughout her entire analysis session. Specifically, we seek to better understand how latency impacts the user's choice of search strategy, or how she decides to navigate her dataset. However, given that the effects of latency can vary widely based on the context of the analysis task, one must study these effects within different contexts. Thus the goal of this study is to evaluate how latency in visualization systems may impact a user's high-level search strategies across different types of data analysis tasks.

2.3.1 Hypotheses

In our study, users explore data in a tile-based format, where the data will be partitioned into fixed-width tiles, similar to interfaces such as Google Maps. We frame our hypotheses in the context of data regions, where a data region is a consecutive block of tiles: some data regions will have tiles with high latency (*i.e.,* will take several seconds to appear in the visualization), and some regions will have low latency tiles (*i.e.,* will appear almost immediately).

Our hypotheses for this study are two-fold. First, we hypothesize that **users prefer to search through low-latency data regions over high-latency ones** when latency is introduced in the system, and have no preference when there is no latency. Second, we hypothesize that **the minimum latency threshold for which users will shift their behavior will be different for analysis tasks** where the user has expert information and where the user has no information (i.e., a novice user).

2.3.2 Participants

We ran our experiments using Amazon Mechanical Turk. 858 people participated, and across all experiments 692 successfully completed the main task. Workers were paid up to $2.27 for completing the study.

2.3.3 Experiment Setup

Our experiments were run using a between-subjects design. Each experiment had five groups of roughly 20 participants, one per latency condition tested: 0, 2.5, 7, 10, and 14 seconds (all experiments evaluate using the same set of latency values). Each participant completed their assigned experiment exactly once with the given delay condition. They also provided consent through a digital consent form, completed a demographics questionnaire, read the instructions for an image-based search task, completed said search task using a browser-based visual exploration tool, and filled out a feedback survey about the task.

Workers were warned that anyone who went through the task too quickly (*i.e.,* ignored the instructions) or too slowly (*i.e.,* took a long break during the task) would have their submission rejected, allowing us to filter out "Bad" datapoints. Furthermore, only participants who successfully completed the task were considered in our evaluation.[1]

Visual Search Task

All of our experiments require participants to complete a visual search task: to explore a grid of bird images (hereafter denoted a *collage*), and to locate one or more target images within this collage that stand out from the rest. In this section, we explain how we designed each collage, and how we created and inserted target images.

Layout: For all but one of our experiments, we created each collage as a grid of 20 images by 20 images (400 total unique images), where each image was 500 pixels by 350 pixels in size. The last experiment used a 200 by 200 grid of images (40,000 total images, where

[1] In the case of the large-scale exploration experiment, all results were reviewed manually to ensure that only workers who made a best effort on the task were included in the evaluation.

(a) The target image (a dinosaur) is featured in the center of the viewport.

(b) Annotated target image used in the Opportunistic search condition.

Figure 2-1: A snapshot of the user study interface. Participants can pan within the viewport window using the mouse.

duplicates are allowed). Each grid consists of a randomized sample from a collection of 1082 bird images. A separate sample of images was selected for each participant. The grid layout is created before the participant starts the task. The exact layout and images used for each experiment and participant were recorded. We refer to each image within the grid as a *tile*.

Target Images: Each experiment includes two or more "target" images that are inserted into the collage. We created two kinds of target images. One set of targets utilizes multiple copies of an image that is completely different from the rest of the collage: a shot of a brown T-Rex dinosaur standing in a forest (shown in the center of Figure 2-1a). The other kind of target images are augmented bird images that are already within the collage. These target bird images were made to stand out from other images in the collage by adding colorful circles to the corners of the images (example shown in Figure 2-1b) .

Interface

The study was run using a browser-based visual exploration interface. The main component of the interface is a 700 pixel by 700 pixel viewport (see Figure 2-1a), which shows only a

portion of the entire image collage to be visible at a time. A user of the interface can explore various parts of the collage by panning to these regions within the viewport. Users can pan in any direction using mouse drag interactions similar to that of Google Maps. No zooming interactions are supported in the interface. As users pan to a new region in the collage, this area first appears blank, showing a grey background. Then image tiles eventually appear after the user finishes their panning interaction. Some amount of delay (0 seconds, up to 14 seconds depending on the latency condition of the experiment) is inserted before each new tile appears. Whenever a user moves away from a region, any tiles that move outside of the viewport are removed. Thus, if the user pans back to this location in the collage, they have to wait for the tiles to re-appear[2].

The interface also includes two function buttons called "FOUND" and "FINISH". When a participant finds a copy of the target picture, they move the target picture to the center of the viewport (*i.e.,* visualization window), and click on the "FOUND" button. The web page then pops out a window to confirm whether the participant did in fact find a target image. When a target is found, the target image is clearly marked with blue text saying "FOUND" after the confirmation, and a running total of "Targets found" is incremented by one. Participants are able to leave the main task page for the user study at any time by clicking the "FINISH" button. Once participants confirmed that they wanted to finish, they were not allowed to interact with the collage anymore.

Evaluating Search Patterns

The primary objective of this study is to evaluate whether people change their search patterns when faced with latency in the visualization interface. Specifically, we aim to measure whether users bias their search behavior when we insert delays into the interface before making images appear in the collage. Examples of how a person may shift their search patterns may include favoring certain regions of the collage over others, or avoiding certain interactions (*e.g.,* panning left vs. right).

Our key evaluation is through the placement of our target images, and how we insert

[2]We note that a common optimization technique for systems like Google Maps is to cache recently visited image tiles. We chose not to simulate any optimization techniques in our study, to ensure that the latencies observed were consistent across participants and image tiles.

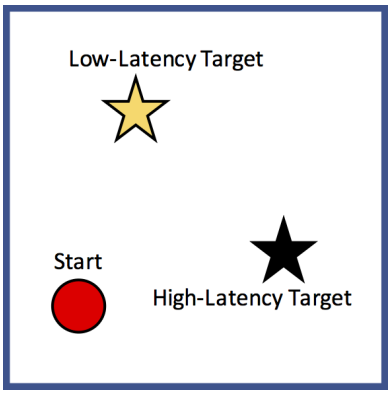


Figure 2-2: A diagram representing a participant's starting position (red circle), and the positions of the low latency target (gold star) and the high latency target (black star).

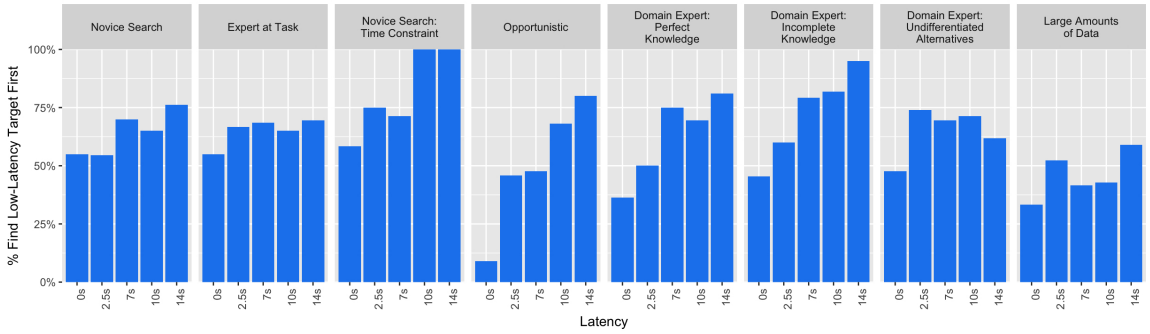


Figure 2-3: Proportion of participants who found the low-latency target first for each experimental condition and latency case.

latency for new images in relation to these target images. In our experimental design, we created two targets, which we will refer to as the low-latency (or fast) target and high-latency (or slow) target. Figure 2-2 is a diagram of an example layout that was used for one of our experimental conditions, showing the participant's starting location within the collage, and the positions of the low-latency and high-latency targets. Suppose that images along the path to the low-latency target load faster than images that are along the path to the high-latency target, our hypothesis is that participants are more likely to find the low-latency target before finding the high-latency target. This hypothesis stems from the outcomes observed in previous work: people notice and become annoyed with delays in visual exploration interfaces [46]. Given a decision of two directions to go in, and experiencing delays along one of those directions, we assume that the user is more likely to choose the direction of least resistance. In contrast, in an environment that is free of laten-

cies towards both targets (*i.e.,* has no noticeable latency), we expect that participants will choose both directions with equal probability (*i.e.,* 50% of the participants will choose the low-latency target and the other 50% will choose the high-latency target).

However, it is possible that participants may be more likely to find a target within a specific region of the collage, simply because they are used to scanning information in a particular direction (*e.g.,* scanning from top to bottom, and from left to right). To take this into account in our study, we alternate which target position is the low-latency position, and which target is the high-latency position. In this way, any effects that may be contributed by positioning are distributed between the two targets.

Introducing Latency

A critical component to our experimental design is carefully controlling how, when, and where latency is introduced as the user explores the collage. Each tile in the collage is assigned a *latency value* that is used to determine the amount of time before the image in the tile appears on the screen. A tile's latency value is based on its relative position to the low-latency and high-latency targets and the participant's current (viewport) location in the collage. We conducted a pilot study to determine suitable latency values, and found the that a latency values ranging from 0 seconds to 14 seconds to be appropriate. Specifically, a tile can have one of four possible latency values: 0 second, 0.75 seconds, 1.5 seconds and n seconds. The value of n is set according to the experimental condition (in our experiments, the possible values are 0 second, 2.5 seconds, 7 seconds, 10 seconds, or 14 seconds. See Section 2.4 for more detail). A tile is assigned n seconds of latency if it is between the user's current location and the high-latency target, 0 seconds if it is between the user's current location and the low-latency target. For all other tiles, its latency value is determined based on whether it is closer to the low-latency or high-latency target. If it is closer to the low-latency target, it is assigned a latency value of 0.75 seconds, and 1.5 seconds otherwise[3].

Because a tile's latency value depends on the user's location, its latency is updated after every user interaction. In our experiment, we "insert" a delay for a particular image to appear on the screen using the `setTimeout` function in Javascript. This function allows

[3]Note that when $n = 0$, all latencies are set to 0

us to set a wait time in milliseconds before we render the image. Each image is given its own separate `setTimeout` call, so that delays can be applied at the granularity of individual images. This wait time is set with respect to the user's current interaction, but is not a blocking function. Therefore, the user is free to perform other panning interactions while waiting for new images to appear.

Data Collection

As participants explore the collage, we record each of their interactions with the interface. For each participant and experiment, we record the same set of information: 1) the initial starting state and parameters for the collage and interface (grid layout, target positions, participant starting position, latency values); 2) the beginning and end positions of each drag interaction, along with their corresponding timestamps; 3) the timestamp and location for every "FOUND" button click within the collage; 4) the timestamp, position, and assigned delay value for each tile that was scheduled to appear in the collage; 5) the timestamp and position for each tile that was removed from the collage by panning away; and 6) the timestamp for the final "FINISH" button click. For each participant that successfully completes the experiment, we compute whether they find the low-latency target first, or the high-latency target first. Figure 2-3 **Novice Search** shows the percentages of the participants who found the low-latency vs. high-latency targets first under five latency conditions: 0 second, 2.5 seconds, 7 seconds, 10 seconds, and 14 seconds.

2.4 Experimental Conditions and Results

Based on our literature review of the common types of analysis scenarios, we conducted a 8 (analysis scenario) x 5 (amount of latency) factorial design experiment to evaluate the effect of latency on user's analysis behavior. Across all 8 of the tested scenarios, we varied the maximum latency: {0 seconds, 2.5 seconds, 7 seconds, 10 seconds, and 14 seconds.} In the sections below, we introduce each of the 8 analysis scenarios, followed by the results of how the different amounts of latency affect the users' behaviors.

2.4.1 Novice Search

This scenario is our "base" condition. Participants were asked to identify two copies of the target dinosaur image within the 20x20 image collage. These copies correspond to one low-latency target and one high-latency target in our experimental design. Two targets were selected to ensure a clear and consistent pattern of latency across the collage. For example, having a single low-latency target ensures that the participant only observes low latencies when moving towards a specific region of the dataset. Participants' starting positions within the collage, as well as the positions of the targets, were kept the same across all participants. Specifically, the starting position is set to be equal-distance apart from both the low-latency and high-latency target.

Hypotheses: In this condition (*i.e.,* analysis scenario), our goal is to evaluate whether latency alone has any affect on how the participants chose to search the collage for the dinosaur targets.

H1.1 Since the novice has no knowledge of where the two targets are, their initial search will be influenced by the latency in that the higher the latency, the more likely they will tend to start the search towards the low-latency target.

H1.2 Because the novice has no knowledge about where the targets are and no preconceived strategy, their search paths will appear to be random.

Results: As expected, we found that roughly half of the participants who successfully completed the 0 second latency case to find the low-latency target first (11 of 20 participants). Thus both targets were equally likely to be found first when there were no delays. However, we also found that in the 2.5 second latency case, both target images were still equally likely to be found first (12 of 22 participants found the low-latency target first). Thus participants' search strategies were unaffected by latency of length 2.5 seconds in the interface. We noticed that higher latencies had a modest effect, the strongest being in the 14 second latency case (16 of 21 participants). To evaluate further, we ran a one-way ANOVA with latency being the independent variable, and number of people who find the low-latency target first as the dependent variable. However we did not find a statistically significant result ($F(4,98) = 0.794$, $p = 0.532$). In summary, we found that latency had little

if no effect on participants' search strategies.

2.4.2 Novice Search Under Time Pressure

In this experiment, we examine how latency affects a user's search behavior when under time pressure. The experimental setup of this condition is the same with the previous, except that the tasks had a timer and were asked to find one of the two targets. The participants were required to complete the task before the timer expires. We ran a pilot experiment to measure the average amount of time it took for participants to successfully find one of the two targets, and determined that 100 seconds was appropriate.

Hypothesis: We stipulate that this task is stressful and difficult to complete because we are asking the participants to find one target within a time frame that makes it extremely difficult search the entire collage.

H2 Due to the added time pressure, the participants are more likely to frantically and randomly search for the targets. As a result, their search behavior will be significantly influenced by latency.

Results: 12 of 21 participants successfully completed the task in the 0 second latency case, 12 of 22 in the 2.5 second latency case, 7 of 21 in the 7 second case, 8 of 21 in the 10 second case, and 9 of 21 in the 14 second case. We observed the general trend of an increase in the fraction of participants that found the low-latency target first as the latency increased, similar to our results from the Novice Search experiment. For latency cases 10 and 14 seconds, all participants found the low-latency target first. We ran a one-way ANOVA with latency as the independent variable, and number of people who find the low-latency target first as the dependent variable. We observed weak significance in the result ($F(4,43) = 2.154$, $p < 0.1$). In summary, we found that high latency (*i.e.,* latency of 10 seconds or more) has a stronger effect on participants who are time pressured, compared to participants who are not time pressured, but low latency (*i.e.,* 7 seconds or less) has no effect.

2.4.3 Analysis by a Technical Expert

In this experiment, we simulate the condition where the user has expert knowledge about the tool itself by revealing that there are some actions that can lead to additional latency in the system.

We do so by adding a note in the study saying: "some actions may be slower than others in the interface." To ensure that the participant is aware of this information, this note is repeated in 3 places: 1) in the task instructions page, 2) as a popup that users had to see before moving on to the task page, and 3) as a short note at the top of the task page.

Hypothesis: The goal of this experiment is to see if users deviate from the behavior we observed in the Novice Search experiment when given information about the interface.

H3 With the additional information about latency in the system, the participant will be more keenly aware of their actions, therefore they will be more likely to avoid searching in places that cause high latency. In effect, the participants' behavior will be affected by the amount of latency.

Results: Ultimately, we found that knowing that some interactions may be slow (*i.e.,* high-latency) seemed to have no effect on participants. We attribute this to participants not realizing that there was a pattern to the slow interactions, and thus that they could potentially choose low-latency (*i.e.,* faster) interactions. In the 0 second latency case, 11 of 20 participants found the low-latency target first, similar to the Novice experiment. We observed only a modest increase as we varied latency, with the highest fraction of people finding the low-latency target first in the 14 second case (16 of 23 participants). We ran a one-way ANOVA with latency as the independent variable and number of people who find the low-latency target first as the dependent variable, and found the result was not statistically significant ($F(4,95) = 0.291$, $p = 0.883$). What we can glean from this experiment is that technical expertise is of limited use on its own: simply knowing that a certain effect may occur in the interface will not tell you *why* it occurs, or what actions to take to prevent the effect from happening. In summary, we found that being made aware of latency in the system had little effect on participants, and similar to the Novice Search condition, latency

had little to no effect on participants.

2.4.4 Analysis by a Domain Expert with Exact Information

The goal of this experiment is to simulate the case where a user may have extensive knowledge about the dataset that helps them navigate the data more efficiently in a visual exploration interface. For example, knowledge of geography helps users significantly when searching for things within geospatial datasets: users can avoid looking in the ocean when searching for something on land, and can pan directly to countries they are familiar with in a world map.

To recreate this condition for our experiment, we provided users with explicit knowledge of the location of the target images. We modified the Novice Search setup by adding a note to the study saying: "The target pictures are directly to the left and right of your starting position." This note was added in 3 places: 1) in the instructions (which included a diagram to help demonstrate the target locations), 2) as a popup that users had to see before moving on to the task, and 3) on the task page. The targets were placed directly to the left and right of the participant's starting location in the collage.

Hypothesis: In this case, participants have nearly perfect information about where two targets are located in the collage.

H4 Because participants know exactly where to explore to find a target, we expect participants to choose more direct or opportunistic search strategies (i.e., look in the specific areas where targets should be found) over exhaustive ones (i.e., look at every image in the collage). Furthermore, tiles along the path to the low-latency target will appear faster than the high latency target, which the participant will see as soon as they start the experiment. Therefore, we should see more participants choosing to explore the part of the collage that appears first, which corresponds to finding the low-latency target over the high-latency target.

Results: We continue to see a general trend that more people find the low-latency target first as the delay length is increased[4]. 8 of 22 participants find the low-latency target first in the 0

[4]The data presented in the 14s case was collected from a second run of the experiment. The original data

second case, up to 17 of 21 participants in the 14 second case. Thus, latency seems to have a noticeable effect on participants' strategies that becomes stronger as latency is increased. To evaluate further, we ran a one-way ANOVA with latency as the independent variable and number of people who find the low-latency target first as the dependent variable, and found the result to be statistically significant ($F(4,101) = 3.402$, $p < 0.05$). With a Tukey post-hoc analysis with a 95% family-wise confidence interval, we found one latency pairing to be statistically significant, 0s vs 14s ($p < 0.05$), and one pairing with weak significance, 0s vs 7s ($p < 0.1$).

2.4.5 Analysis with Incomplete Domain Knowledge

The goal of this experiment is to simulate the case where a user may have prior knowledge about the dataset, but their knowledge has some uncertainty associated with it: the user knows of the possible locations they may want to search, but they don't know which location is the "correct" one. To create this experiment, we updated the Novice Search design by adding a note to the study saying there is only one target, and it is equally likely that the target is either to the left or to the right of the participant's starting location, similar to the previous experiment. This note appeared in the same locations in the task website as the previous experiment. The targets were placed directly to the left and right of the participant's starting location.

Hypothesis: In this experiment, participants have extensive information about the target location (*i.e.,* the target is either on the left, or the right). However the target only appears to be on one side, so they must still choose which direction in which they want to search.

H5 Given the target location information, participants will be more likely to be influenced by latency, and will heavily prefer to find the low-latency target first.

Results: The fraction of people who found the low-latency target first was similar in the 0 second latency case (8 of 20 participants) to the results observed for the Novice Search experiment (11 of 20 participants). We also see a steady increase in the fraction of people who found the low-latency target first when the latency was increased, with 19 of 21 par-

was omitted due to issues with the first experimental run.

ticipants finding the low-latency target first in the 14 second case. To investigate, we ran a one-way ANOVA with latency as the independent variable, and number of people who find the low-latency target first as the dependent variable. We found the result to be statistically significant ($F(4,104) = 3.892$, $p < 0.01$). We ran a follow-up analysis using Tukey post-hoc analysis with a 95% family-wise confidence interval, and found two pairs to be statistically significant: 0s vs 10s ($p < 0.05$) and 0s vs 14s ($p < 0.01$). We found one pair with weak significance: 0s vs 7s ($p < 0.1$).

2.4.6 Analysis with Tacit Domain Knowledge

The goal of this experiment is to simulate the case where a user may have prior knowledge about the dataset that helps them better navigate the data, but their knowledge is vague: they know generally where they want to go, but not an exact location.

To simulate this condition in the experiment, we updated the Novice Search setup by adding a note to the study saying there is only one target, and it is somewhere on the left-hand side of the collage. This note appeared in the same location as the previous experiment. The targets were placed in the top right and bottom left sides of the *left half* of the collage.

Hypothesis: Participants will be able to eliminate half of the collage for the search, since the target must be on the left-hand side. As such, our hypothesis is as follows:

H6 We expect participants to perform a more direct search towards the target. Given that participants do not know the exact location, we expect for them to still be influenced by latency in terms of the general direction of their search.

Results: In this experiment, the fraction of people who found the low-latency target first for the 0 second case (10 of 21 participants) was similar to the results observed for the Novice Search experiment (11 of 20 participants). However, we found a noticeable increase in the fraction of people who found the low-latency target first when the delay was increased to 2.5 seconds (17 of 23). However, we do not see a a clear trend in the fraction of people finding the low-latency target first as we vary latency beyond 2.5 seconds. In a one-way ANOVA,

we did not find a significant F measure ($F(4,104) = 0.2287$, $p = 0.379$). In summary, we found that even though participants could eliminate half of the collage during their search, the search space was still to broad for participants to feel comfortable switching to a more direct or opportunistic search strategy. As such, our results for this experiment are similar to the results for Novice Search.

2.4.7 Opportunistic Search and Analysis

In our previous experiments, we found that people were unlikely to deviate from a structured search strategy given the layout of the collage (i.e., gridded layout), and the "needle in a haystack" style search participants were asked to perform (except for the Analysis by a Domain Expert and Analysis with Incomplete Domain Knowledge experiments). With limited information about the dataset as a whole, participants were unwilling to deviate from their basic strategies, which we interpreted as there being too much risk involved in performing a random search over an unknown dataset. To design an experimental condition that simulated a dataset that provided more opportunities for exploration, we created 54 total targets in the collage (27 orange targets, and 27 purple targets), compared to the 2 targets used in previous experiments. The collage was divided into top and bottom halves. The low-latency half had 21 orange targets and 6 purple targets spread across it, and 21 purple targets and 6 orange targets on the high-latency half. Participants were asked to find 18 targets within the collage. Since so many targets were available on each half of the collage, the goal behind the design was to lead the participant along one half of the collage to find targets.

Given the abundance of targets in this case, image tiles were only given one of two latencies: "fast" or "slow", corresponding to the best (0 second) and worst (n seconds) latency parameters, respectively. The low-latency half had no latency (0 sec), and the high-latency half had uniform latency (0sec, 2.5sec, 7sec, 10sec, or 14sec, depending on n). Rather than using dinosaurs, each target was a modified bird picture. Specifically, a bird picture became a "target" picture by marking it with either orange circles or purple circles in the corners. Low-latency targets are orange, and high-latency targets are purple, however

no distinction is made between the different target colors in the experiment, so participants were free to find and select targets of either color.

The starting position was along the left border, in the middle of the y axis of the collage. One low-latency target and one high-latency target were partially visible within the viewport window, so the user could see the colorful circles. However since longer latencies are associated with the high-latency half of the collage, the high-latency target took longer to load for the latency cases of 2.5 seconds or longer.

Hypothesis: In this experiment, a large number of targets are spread throughout the collage, so participants have the ability to find a large number of targets within a small region of the collage.

H7 Given that the low-latency half of the collage will appear before the high-latency half, participants will be drawn to the visual cues associated with the low-latency targets. With the abundance of targets, participants will consistently prefer to continue their search on the low-latency half of the collage (i.e., participants will show a heavy influence from latency).

Results: Given the large number of targets participants were asked to find, we had to develop a new strategy for calculating whether participants found the low-latency targets first. To do this, we computed the fraction of orange targets that were found by each participant, and labeled the participant as having found the low-latency targets first if more than 78% of the targets they found were orange targets (*i.e.,* 14 or more orange targets found). 78% was chosen to match the distribution of targets on each half of the collage: 21 of the 27 targets on the low-latency half of the collage were low-latency targets, and 21 of 27 targets on the high-latency half were high-latency targets. The experiment was designed such that the only way participants could find this many low-latency targets was by performing the vast majority of their exploration on the low-latency half of the collage.

We found that in the 0 second latency case, very few participants explored the low-latency half first (2 of 22). Many participants in the 0 latency case had about half of their targets found being low-latency targets (i.e., 8-11 of targets), often because they explored both halves of the collage nearly equally. However, we see a clear increase in the fraction of

people finding the low-latency targets first as we increase latency. In the 14 second latency case, we see that 80% of participants who experienced these latencies chose to explore the low-latency half of the collage (20 of 25), showing a clear preference for the half of the collage with lower latencies, compared to the 0 second latency case.

We followed up our analysis with a one-way ANOVA using latency as the independent variable, and number of people who found a high fraction of low-latency targets as the dependent variable. We observed an F measure of $F(4,109) = 8.385$ ($p < 0.0001$). In a Tukey post-hoc analysis with 95% family-wise confidence interval, we found 4 statistically significant latency pairings: 0s vs 2.5s ($p < 0.05$), 0s vs 7s ($p < 0.05$), 0s vs 10s ($p < 0.001$), 0s vs 14s ($p < 0.0001$). We also saw weak significance for the pair 2.5s vs 14s ($p < 0.1$).

2.4.8 Analysis of Large Amounts of Data

The goal of this experiment is to simulate the condition where the underlying dataset is simply too large to explore everything in a single session, a case that is quickly becoming the norm for visual data exploration. As such, this experiment is designed to make it extremely difficult to successfully execute a full scan of the entire dataset. For this experiment, we updated the Novice Search setup by making the grid size significantly larger: from 20 images by 20 images to 200 by 200 images, where repeats are allowed in the collage. Two dinosaur target images are still inserted in the collage, similar to previous experiments. To warn participants of the size of the collage, we added a note to the study that emphasized the following points: 1) there are 40,000 images in the collage, 2) it is infeasible to scan everything in a reasonable amount of time, and 3) they should be creative and try a different approach. This note appeared in the same locations in the task website as the previous experiment.

Hypothesis: The collage is too large for participants to successfully perform an exhaustive search, especially without prior knowledge.

H8 Because the collage is nearly impossible to search exhaustively, participants will adopt a random search strategy that will also be affected by latency (similar to the Novice Search experiment).

Results: We found that across all five latency cases, only four participants were able to successfully complete the task (4 of 106). Given the low completion rate, and the massive size of the collage, we could not rely on our original techniques for comparing participants. Therefore, we took a different approach to analyzing the data. For participants that did not find either target, we instead considered whether they performed over 78% of their panning interactions on the low-latency half of the collage (i.e., on the same side as the low-latency target). We chose this percentage to be consistent with our Opportunistic Search condition, which also involves analyzing whether participants explore a particular half of the collage. We found an increase in the fraction of people that explored the low-latency half of the collage when delays were applied: from 7 of 21 participants in the 0 second case, compared with 13 of 22 participants in the 14 seconds case (only a marginal increase was encountered in the other latency cases).

We followed up on this analysis with a one-way ANOVA with latency as the independent variable, and number of people who explore the low-latency half of the collage as the dependent variable, however we did not find these results to be statistically significant ($F(4,104) = 0.859$, $p = 0.492$). These results are consistent with our findings in the Novice Search case (*i.e.,* that latency has little effect on user behavior), which makes sense given that this experiment is essentially the Novice Search condition on a massive collage.

2.4.9 Summary of Results

In total, we analyzed 8 separate analysis scenarios: **Novice Search** (Section 2.4.1), **Novice Search Under Time Pressure** (Section 2.4.2), **Analysis by a Technical Expert** (Section 2.4.3), **Analysis by a Domain Expert** (Section 2.4.4), **Analysis with Incomplete Domain Knowledge** (*i.e.,* undifferentiated alternatives, Section 2.4.5), **Analysis with Tacit Domain Knowledge** (Section 2.4.6), **Opportunistic Search** (Section 2.4.7), and **Analysis of Large Amounts of Data** (Section 2.4.8). The following analysis scenarios did *not* have significant results: Novice Search, Novice Search Under Time Pressure, Analysis by a Technical Expert, Analysis with Tacit Domain Knowledge, and Analysis of Large Amounts of Data. In four out of five of these experiments, participants had no information

about the locations of the targets. In the Tacit Domain Knowledge experiment, participants only had vague knowledge about the locations of the targets. We found statistically significant results for the remaining three experiments: Analysis by a Domain Expert, Analysis with Incomplete Domain Knowledge, and Opportunistic Search. In all three of these experiments, participants had a considerable amount of information about the location of the targets. Thus, participants seem to shift their behavior when given more information about the dataset itself. We report results for a global analysis across all analysis scenarios in Section 2.5, and discuss our statistical results in more detail in Section 2.7.

2.5 Statistical Analysis

Our main hypothesis is that people behave differently during visual search tasks when they encounter latency in the interface, compared to when they search in a no-latency environment. To test this hypothesis, we designed our experiment to have high latency and low latency regions in the collage. We asked participants to search for strategically placed target images within the collage (i.e., the low-latency and high-latency targets), and made panning interactions towards these targets respond faster (*i.e.,* lower latency) and slower (*i.e.,* higher latency), respectively. Thus, in the context of the experiment, our hypothesis becomes that participants are more likely to find the low-latency target first (*i.e.,* prefer to search low-latency regions), rather than finding the high-latency target first (i.e., prefer to search high-latency regions), when there is latency in the system. In the control condition (i.e., no latency), we would expect the two targets to be found equally.

To evaluate how different perceptions and environmental conditions may affect participants' perceptions of latency, we also tested 7 different variations of our original experiment. A secondary hypothesis is that certain environmental conditions affect the "potency" of the latencies. In particular, we hypothesize that learning more information about the underlying dataset, either beforehand (*i.e.,* search with domain expertise) or as one explores (*i.e.,* opportunistic search), makes the user more susceptible to the effects of latency.

To evaluate our experimental results, we ran a two-way ANOVA to test for meaningful differences between our independent variables: visualization condition (i.e., experimental

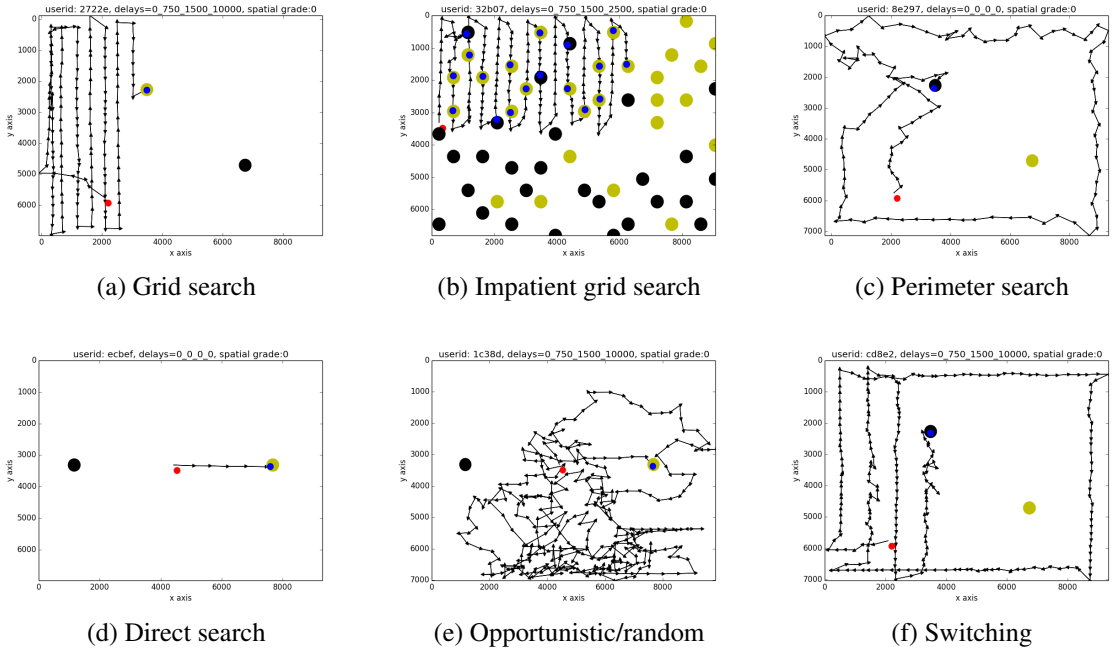
condition) and latency. Seven out of eight experiments included the visualization condition as a variable (the Exploring Large Amounts of Data experiment was omitted due to lack of successful task completions). The latency variable included 5 values, one per latency case measured (0, 2.5, 7, 10, and 14 seconds). Our dependent variable was the number of participants who found the low-latency target first. We found statistically significant main effects for both visualization condition ($F(6,654) = 2.940$, $p < 0.01$) and latency ($F(4,654) = 12.983$, $p < 0.0001$). We did not find the interaction between the two variables to be statistically significant ($F(24,654) = 0.949$, $p = 0.533$).

We performed a Tukey post-hoc analysis for visualization condition with 95% family-wise confidence interval, and found two pairs of visualization conditions to be statistically significant: "Domain Knowledge: Incomplete Information" vs. "Opportunistic Search" ($p < 0.05$) and "Novice Search Under Time Pressure" vs. "Opportunistic Search" ($p < 0.01$). We also performed a Tukey post-hoc analysis for latency with 95% family-wise confidence interval, and found 5 latency pairs to be statistically significant: 0s vs 2.5s ($p < 0.05$), 0s vs 7s ($p < 0.0001$), 0s vs 10s ($p < 0.0001$), 0s vs 14s ($p < 0.0001$), and 2.5s vs 14s ($p < 0.01$).

These results give us a general idea of whether visualization condition and latency impact user behaviors, particularly in the case of latency: we see a clear distinction between the no latency case (i.e., the 0 second case) and all other latency cases (2.5 seconds and higher). However, we do not see a statistically meaningful difference between the majority of visualization condition pairings, nor the latency pairings. Furthermore, we can see in Figure 2-3 that latency seems to have only a minimal effect on user behavior for some experiments (e.g., the Expert of Task experiment), for which we need to dig deeper beyond the statistical analysis. Furthermore, relying strictly on p-values to assert patterns and significance is known to be problematic [28]. Given our relatively small sample size (roughly 20 participants) for each condition and latency case, performing a pairwise statistical analysis becomes more challenging. Therefore, we see this statistical analysis a starting point for better understanding the effects of latency on user search behavior. We found that analyzing the search patterns themselves provides more compelling evidence for how participants apply one or more search strategies in the interface, and more importantly, how their strategies change across experimental condition and worst-case latency value. Our results are

discussed in Section 2.6.

2.6 Analysis of Interaction Trails



(a) Grid search (b) Impatient grid search (c) Perimeter search

(d) Direct search (e) Opportunistic/random (f) Switching

Figure 2-4: Examples of the five types of search strategies identified through the interaction trails: grid search, impatient grid search, perimeter search, direct search, and opportunistic/random search. Figure 2-4f is an example of how participants apply multiple strategies.

In our statistical analysis we found a meaningful difference in user preferences for searching low-latency regions across different visualization conditions, as well as across different worst-case latencies. However, this analysis is unable to tell us what those differences are, and why these differences matter. As such, we decided to inspect the analysis trails of all 692 successful completions, as well as the 102 unsuccessful attempts from the Analysis of Large Amounts of Data experiment (794 total). To perform this analysis, the images were manually clustered by two independent classifiers according to visual similarity of the search trace patterns. 5 high-level patterns emerged from the classification: grid search, impatient grid search, perimeter search, direct search, and random (or opportunistic) search. An example of each search strategy is provided in Figure 2-4.

Grid search (Figure 2-4a) is distinguished by a back-and-forth scan that occurs across the full width of the collage. With impatient grid search (Figure 2-4b), the participant performs a grid search over a much smaller region of the collage. In perimeter search (Fig 2-4c), the participant pans along the perimeter of the collage, gaining familiarity with the boundaries and size of the collage, and looking for an opportune region for panning inward. Direct search (Figure 2-4d) is distinguished by a (nearly) straight path towards one of the targets. Unlike the other search strategies, Random search (Figure 2-4e) is characterized by a seemingly haphazard search path.

We also found that some participants utilized multiple strategies while performing a single search task (example in Figure 2-4f. We analyze these switches between strategies in Section 2.6.2.

2.6.1 Strategies Across Conditions

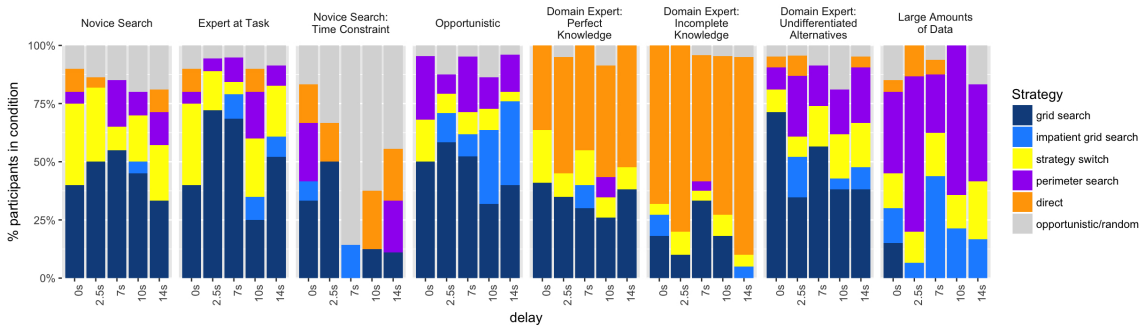


Figure 2-5: Distribution of search strategies across conditions.

We found clear differences in how many people used each of the five strategy types across experimental conditions, shown in Figure 2-5. We see that highly structured strategies like grid and perimeter search dominate in the **Novice Search** and **Expert of Task** experiments. In these experiments, the dimensions of the collage and exact location of the targets are unknown to the participants. Thus participants must work with information that they can glean from the collage itself to execute their search: the gridded structure of the collage. However, when participants are under **time pressure**, we see a significant increase in the fraction of participants that choose to utilize a random or direct search strategy. The

fraction of participants that chose these strategies increased dramatically from the no latency case (2 of 12 participants) to 7 seconds of latency and higher (6 of 9 participants for the 14 seconds case). We also found that none of the participants switched strategies part way through their search. This behavior is consistent with the outcomes observed in prior work (see Section 2.2.2 for examples): people default to simple (and familiar) search strategies when placed under time constraints, even when those same people would select a more effective strategy with the time constraint removed. In the **Opportunistic Search** experiment, we see a significant increase in the fraction of participants that choose an impatient grid search strategy. The fractions are highest for the 10 and 14 second latencies (7 of 22 participants, and 9 of 25 participants, respectively). 21 of 21 participants that chose an impatient grid search restricted their search to the low-latency half of the collage only. Thus latency had a clear effect on how participants executed this strategy, even in the 2.5 second case, where 3 of 3 participants only explored the low-latency half of the collage. In the **Domain Expert: Exact** and **Incomplete Information** experiments, we see a high proportion of people that select a direct search strategy. This is consistent with the experimental design, where participants know the locations of the targets in advance. These results also speak to observations in past work on domain expertise: when people come to the analysis with domain expertise, they tend to employ more efficient search strategies. We find that in the **Undifferentiated Alternatives** experiment, participants revert back to using structured search strategies, such as grid and perimeter search. We attribute this to an insufficient amount of information about the location of the targets: searching half of the grid was still too large of a space for participants to feel confident in choosing a more efficient or more opportunistic search strategy.

2.6.2 Latency and Switching Strategies

Of the 794 completions analyzed, 107 participants utilized more than one distinct pattern in their search trace (692 successful completions, and 102 unsuccessful from the Analysis of Large Amounts of Data experiment). We analyzed which strategies were used initially by participants (i.e. at the start of their search) as well as which strategies they switched to (see

		Delay				
Condition	strategy	0s	2.5s	7s	10s	14s
Novice Search	grid-other				1	2
	other-grid	2	3		1	
	perimeter-grid	5	4	2	2	3
Expert at Task	grid-other				1	1
	other-grid	2			1	1
	perimeter-grid	5	3	1	2	1
Opportunistic	grid-other	3			1	
	perimeter-grid		1			
Domain Expert: Perfect Knowledge	grid-other					1
	other-grid	3		1		1
	other-perimeter	1			1	
	perimeter-grid	1	2		1	
Domain Expert: Incomplete..	grid-other					1
	perimeter-grid	1	2		1	
Domain Expert: Undifferentiated Alternatives	grid-other		1			2
	other-grid	1		1		
	other-perimeter		1		2	1
	perimeter-grid			3		1
Large Amounts of Data	grid-other	2		1		
	other-perimeter	1	2	2	1	3

Figure 2-6: Distribution of strategy switches across conditions. Each row has a pair of strategies "A-B", where A is the initial strategy that was used (*e.g.,* grid search), and B is the final strategy that was used. For example, the third row represents a count of participants that started the task using grid search, and then switched to a perimeter search. "Other" refers to a strategy that is neither grid search nor perimeter search (*e.g.,* direct search).

Fig. 2-6). In the Novice Search and Expert of Task experiments, we find that people tend to start with a perimeter search, then switch to a grid search. In general, participants heavily favored switching to a grid search across all latency cases (22 out of 25 total participants for **Novice Search**, 16 of 18 for **Expert of Task**). In the **Opportunistic** search task, we see a complete shift compared to Novice Search: people start with a perimeter or grid search (11 of 11 participants), but then switch to an alternative search strategy (10 of 11 participants). This points to a shift in how participants interact with the data: initially, they attempt to thoroughly explore the collage. However, when they see the abundance of targets, they switch to a more efficient or opportunistic strategy. In the **Domain Expert: Incomplete**

Information experiment, we found that people tend to start with perimeter search before switching (6 of 7), and 4 of 7 participants switched to a grid search. We hypothesize that these participants used the target location information to direct their search towards a particular half of the collage (via perimeter search), then switched to a different search strategy to find the target. In the **Domain Expert: Exact Information** experiment, we saw yet another change in the distribution: people were split evenly between starting with an alternative search strategy (7 of 14 participants) and starting with a perimeter search (6 of 14). These results are consistent with more participants attempting a more efficient strategy to find the target, compared to the Domain Knowledge: Incomplete Information experiment. The majority of participants eventually switched to a grid search (9 of 14). Thus participants try a more efficient search strategy first with an unsuccessful outcome, and then switch to a more conservative strategy for the rest of the task. In the **Analysis of Large Amounts of Data** experiment, we found that most participants started with an alternative search strategy (9 of 13 participants), and then switched to perimeter search (or attempted to). In the Novice Search experiment, participants relied on the size and structure of the collage to perform a perimeter search or grid search. In this experiment, participants are asked to perform the same task, but in an overwhelmingly large collage. Given the high fraction of participants that choose a perimeter or grid search strategy in similar experiments, we attribute these strategy shifts to participants becoming disoriented in the collage, and seeking a collage boundary to reorient themselves. However, with a massive collage of 40,000 images, many participants never reached the perimeter before calling off the search.

2.7 Discussion

2.7.1 Comparing with Real-World Domain Experience

An important motivator for this work is its relevance to how domain experts explore data in the real world. Our experimental design was influenced by our experiences from working with earth scientists. We found that when we designed user study tasks that leveraged their

existing experience, earth scientists (*i.e.,* our domain experts) utilized similar search strategies to our observations from Section 2.6. In our most prominent example, we recruited earth scientists to search for mountain ranges in a satellite sensor dataset. When we asked earth scientists at UC Santa Barbara and the University of Washington to search for visibly snowy mountain ranges, they immediately panned to the Rocky mountains (*i.e.,* mountains near the west coast). When we asked earth scientists at MIT to perform the same task, they immediately panned to the Appalachian mountains (*i.e.,* mountains near the east coast). The behavior of the earth scientists was very similar to what we observed for the direct search pattern in our study on Amazon Mechanical Turk. Thus we have observed that real domain experts exhibit similar behavior to the Turkers with simulated domain expertise in our analysis scenarios. We plan to extend this work in the future by performing similar studies with real domain experts and real-world data.

2.7.2 Effects of Delay Length on Performance

We found a general trend in the effects of latency on visual search tasks: the longer the worst case latency (i.e., the delay case), the more likely it is that the user will prefer to explore regions with lower latency. When latency was introduced in the interface, we observed an increase in the number of people who identified target images in low-latency data regions. This effect was most pronounced (and statistically significant) when users had more domain knowledge (*e.g.,* knew there were many target images, or knew the location of the targets).

However, we also found that latencies of 2.5 seconds had little or no effect on participants for five out of eight of our experiments. Furthermore, it was generally at the 14 second case where we observed statistically significant deviations in search behavior, when compared to the 0 second case. These findings differ drastically from the outcomes of the latency study by Liu and Heer [62], where they found that delays beyond 500 milliseconds had a clear negative effect on participants, and delays beyond 1 second rendered the interface "unusable" by the participants. We attribute the differences between our study and the Liu and Heer study to two factors: 1) major differences in the implementation of delays in

the interfaces, and 2) people's expectations for the interfaces that were studied.

In the imMens system [63], participants cannot perform subsequent interactions with the interface until existing interaction events have finished. Furthermore, every interaction has an added delay of 500 millisenconds. In this scenario, even a short delay of 500 milliseconds can be frustrating for participants, since every interaction is guaranteed to take at least 500 milliseconds. However, this latency model is inconsistent with other popular pan-zoom interfaces like Google Maps, which employ *asynchronous* loading of image tiles. To provide a realistic exploration environment for participants, our interface design utilizes asynchronous loading of image tiles, where each tile may be given one of four latency values, from 0 seconds up to the worst case latency (2.5 to 14 seconds, see Section 2.3 for details). Furthermore, participants did not have to wait for all tiles to load before performing their next interaction, and thus could continue to make progress on the task, even when some image tiles took a long time to load. We believe this results in very different usage patterns for the resulting interfaces, and ultimately to a weaker latency effect in the asynchronous case.

Another possible factor is that users have different "expectations" for different interfaces. For example, many people are now accustomed to Google Maps and its latency profile. Since the visualization used in our image search tasks shares similarities in the rendering (*i.e.,* using tiles that are loaded asynchronously) and the interaction design (i.e. the use of dragging) with Google Maps, it is reasonable to assume that the participants had expectations of the behavior of the system based on their experience with Google Maps. In Google Maps, depending on network latency, image tiles can appear seconds or tens of seconds after a user's interaction. In contrast, the imMens system utilizes a coordinated view visualization with four supported interaction types, including brushing and linking and selection, and Liu and Heer found that inserting 500 millisecond delays had a significant effect on participants. The difference between these two studies suggest that the interaction designs may play an important factor on our perception and tolerance of latency.

2.7.3 Effects of Explicit Knowledge of Latency

In the case of the Analysis by a Technical Expert experiment, we found that even though participants were made aware that the latencies were associated with interactions, very few participants noticed a pattern to the delays in the interface. Participants seemed to accept that delays were simply a part of the interface, and failed to realize that they could control whether the latencies were low or high by changing the direction of their panning interactions. Unsurprisingly, we observed similar results to our Novice Search Experiment in this case, where neither experiment had statistically significant results (see Section 2.4.9 for an overview of all 8 experiments). This outcome is consistent with our design methodology, as knowledge of the interface is orthogonal to knowledge of the underlying dataset. Furthermore, we have observed that it is with knowledge of the data that participants choose to deviate from structured search strategies (*e.g.,* grid and perimeter search). This idea was reinforced through the results of the other experiments, where participants had expert domain knowledge. The Opportunistic Search experiment was a clear example of this idea, where participants had more information about the data that they could learn as they explored (i.e., more targets to find), and they were able adapt their search strategies accordingly.

This points to an intuitive cost model for search tasks that balances two factors: speed and precision. When knowledge of the data is too uncertain (*e.g.,* seemingly random), people employ search strategies that favor precision over speed. Why focus on picking the fastest (*i.e.,* lowest latency) interactions, when one has no idea whether faster interactions will actually lead to finishing the task faster? In a needle-in-a-haystack style search task, overlooking a particular region could easily lead to failure. Furthermore, we found in our experiments that eliminating half of the collage (i.e., going from finding 1 target in 400 images to 1 in 200 images) was still not precise enough for people to be persuaded to favor speed over precision. However, when more precise information is provided about the dataset (e.g., the location of the targets), we see a shift in participants' behavior. They make more decisions that favor speed (*i.e.,* lower latency interactions) over efficiency (*i.e.,* comprehensive search).

2.8 Summary

In this chapter, we evaluated the effect of latency in a visualization on the user's behavior in an image search task. In particular, we examined how the effect of latency differs in 8 analysis scenarios: (1) analysis by a novice analyst, (2) analysis under time pressure, (3) analysis by an expert of the task, (4) analysis by an expert of the data, (5) analysis with incomplete information about the data, (6) analysis by an expert with tacit Domain knowledge, (7) opportunistic search and analysis, and (8) analysis of large amounts of data. Through an 8 (analysis scenario) x 5 (amount of latency) factorial design study, we show how latency affects users' search behaviors in the context of other factors such as task difficulty and familiarity with the data. Our findings show that latency does not affect all scenarios the same way. In some cases, such as opportunistic search, latency can significantly affect the user's behavior. However, in other scenarios such as analysis by an expert with tacit domain knowledge, the effect of latency is not noticeable. In addition, our analysis of the user's search paths reveals 5 different search strategies that people employ under different task and latency conditions. Furthermore, we found that participants actively switch between different search strategies as latency in the system increases, such as adopting a random strategy when under time pressure, or adopting a more direct search strategy when knowledgeable of the underlying dataset.

Chapter 3

The Sculpin System

3.1 Introduction

Through our analysis of system latency in the previous chapter, we suggested a relationship between high latencies and shifts in people's preferred search strategies. From these behavioral shifts, we found that latency has a clear effect on how people choose to navigate a visualization when performing search-based analysis tasks. This effect becomes more pronounced when we have expert users with domain knowledge (*i.e.,* knowledge of the underlying dataset), such as data scientists. One common interaction pattern we have observed from data scientists is that they analyze a small region within a larger dataset, and then pan to a nearby region and repeat the same analysis. They initially aggregate or sample these regions when looking for a quick answer, and zoom into the data when an exact answer is needed. Thus, we focus on supporting a detail-on-demand browsing paradigm, where users can pan to different regions within a single dataset, and zoom into these regions to see them in greater detail.

Following from the general latency analysis in the previous chapter, we studied how latency impacts the performance of visualization tools designed for browsing massive datasets (or *exploratory browsing*). A diagram of the standard exploratory browsing architecture is provided in Chapter 1 (Figure 1-1). Here, the user interacts with a visualization tool running on a client machine (*i.e.,* the user's laptop), and the client is connected to a DBMS running on a remote server. The user formulates a DBMS query to explore, then

the query is sent to and executed on the DBMS, and the results are sent back to the client. The client produces a visualization of these query results, and the user interacts with the visualization through a detail-on-demand, or pan-zoom interface. The user can pan to various regions in the query results and zoom to see these regions in more or less detail. After interacting with this visualization, the user can choose to write a new query to visually explore a different facet of the underlying dataset. Thus the exploration process is not only interactive but also iterative in nature, as the user continues to write queries and explore her data.

A major challenge in exploratory browsing lies in reducing overall system latency. While users want to be able to drill down into specific regions of a dataset, they also want their actions within the browsing tool to be fluid and interactive. Even one second of delay after a pan or zoom can be frustrating for users, hindering their analyses and distracting them from what the data has to offer [72, 62]. Although modern database management systems (DBMS's) allow users to perform complex scientific analyses over large datasets [81], DBMS's are not designed to respond to queries at interactive speeds [10, 48], resulting in long interaction delays for browsing tools that must wait for answers from a backend DBMS. Thus, new optimization techniques are needed to address the non-interactive performance of modern DBMS's, within the context of exploratory browsing.

We observe that there are two types of latency in exploratory visualization systems that must be addressed. First, there are the "interaction latencies", which are delays in a system's response to a user's interactions during the exploration process (e.g., latency in a panning or zooming interaction). Second, there are the "materialization latencies", which represent the amount of time it takes for the system to prepare the data for exploration, including the time to execute the user's DBMS query (see Figure 3-1). While there have been a good number of exploratory browsing systems developed for reducing interaction latencies (*e.g.,* [9, 63, 17, 60, 76]), we have found no systems that adequately address the issue of materialization latency.

Materialization latency is an important, but often overlooked part of data exploration. In most systems that address interaction latency, the systems assume that data preparation, or query materialization, is an offline task that imposes no cost to the exploration

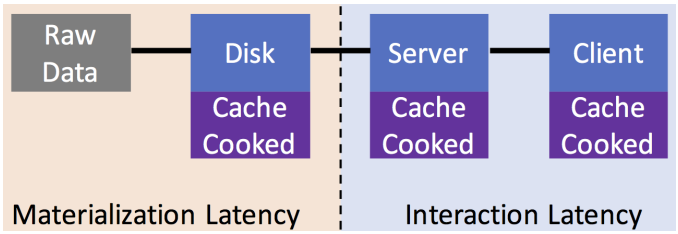


Figure 3-1: Storage layout for caching materialized query results (or "cooked" data), from left to right (raw input data is considered separately): 1) server disk, 2) server main memory, and 3) client main memory. To reduce interaction latency, existing systems only cache query results on the client [17, 63, 60, 76]. None of these systems consider materialization latency.

process. However, this assumption is often not true. For example, in large data exploration, exploratory systems that use precomputed data cubes such as imMens [63] and NanoCubes [60] need to re-compute the data cubes if the user wishes to explore data dimensions that are not part of the data cubes. Similarly, sampling-based systems such as BlinkDB [4] and Sample-and-Seek [25] would require building new samples if the user's inquiries significantly differ from the workloads used to build the initial stratified samples. With progressive sampling [47, 23, 33], the user still has to wait for her queries to complete to see accurate results. Even for systems that use predictive pre-fetching [17, 27, 26], data preparation is needed for this technique to be effective. In effect, ignoring the cost of materialization latency assumes a limited context of a user's data exploration process.

3.1.1 Sculpin

In this thesis, we propose a new exploratory browsing system called Sculpin to support interactive exploration of massive arrays. Sculpin takes into account both materialization latency and interaction latency, and to the best of our knowledge, is the first to optimize across the entire exploratory browsing architecture. To reduce the time and space spent materializing query results, Sculpin applies resource-conscious materialization (or pre-computation) techniques. To reduce interaction latency, Sculpin combines data caching and pre-fetching across all three layers of storage (see Figure 3-1: main memory on the client, main memory on the server, and server disk.

When the user performs zooms in Sculpin, she expects to see more detail from the underlying data. To support multiple levels of detail, we insert aggregation operations into

the user's query. However, complex scientific analyses take time, and may not execute at interactive speeds in the DBMS. To ensure that zooms are fast in Sculpin, we compute a subset of *zoom levels*, or levels of detail, beforehand, and store them on disk. Each zoom level is treated as a separate materialized view, which we can partition into equal-size blocks, or *data tiles* [63], allowing for efficient data retrieval by the client.

To support exploration along multiple dimensions, Sculpin creates an ensemble of "navigable" visualizations, often referred to as coordinate multiple-view visualizations (CMV) [86], where each visualization maps to specific dimensions in the underlying dataset. In CMV, the user can pan or zoom using a particular visualization to explore along the corresponding dimensions.

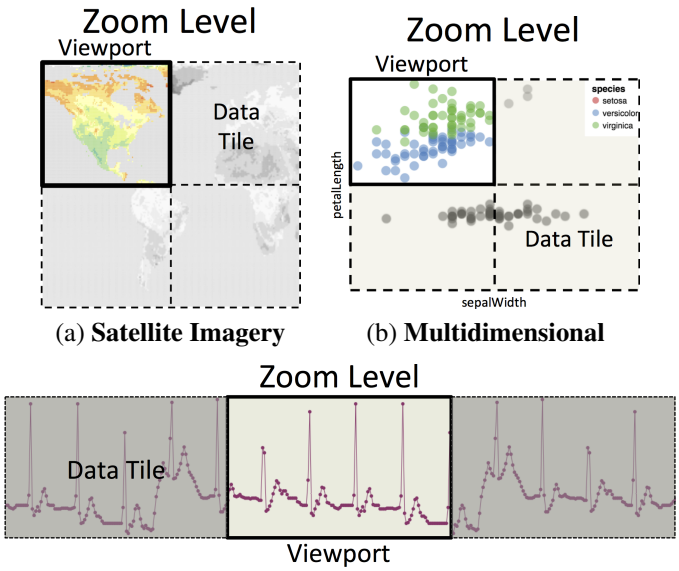
In this chapter, we present Sculpin's tile-based data model for arrays, and general-purpose architecture for supporting exploration of data tiles using the array-based DBMS SciDB [94]. The subsequent chapters explain the optimization techniques implemented in Sculpin to enable exploration of massive array data at interactive speeds.

3.2 Data Model

Sculpin is designed to support exploration of massive arrays (*i.e.,* data represented in a matrix-based format). In this section, we describe the kinds of array data supported by Sculpin, and our process for building zoom levels and data tiles.

3.2.1 Datasets Supported by Sculpin

The datasets that work best with Sculpin share the same properties that make SciDB performant: (a) the majority of column types are numerical (integers, floats, etc.), and (b) the relative position of points within these columns matters (*e.g.,* comparing points in time, or in latitude-longitude position). These properties ensure that the underlying arrays are straightforward to aggregate, partition, and visualize in Sculpin. The following three example datasets share these properties: geospatial data (*e.g.,* satellite imagery in Figure 3-2a), multidimensional data (*e.g.,* iris flower classification in Figure 3-2b), and time series data (*e.g.,* heart rate monitoring in Figure 3-2c). Beyond these three examples, SciDB has also



(a) **Satellite Imagery** (b) **Multidimensional**

(c) **Timeseries (Heart rate Monitoring)**

Figure 3-2: Potential tiling schemes for three types of data.

been used for efficiently analyzing genomics data [96] and astronomy data [93]. Given its extensive support for complex analytics over multidimensional datasets, we use SciDB as the back-end DBMS in Sculpin.

Consider Figure 3-2a, where the user is exploring an array of snow cover measurements computed from satellite imagery. Each array cell has been mapped to a pixel, where orange and yellow pixels correspond to snow. We have partitioned the current zoom level along the array's two dimensions (latitude and longitude), resulting in four data tiles. The user's current viewport is located at the top left data tile; the user can move to other tiles by panning in the client-side interface. The user can also zoom in or out to explore different zoom levels. In Figure 3-2b, a multidimensional dataset is being explored along two dimensions: petalLength and sepalWidth. Since the user is only exploring two of the total dimensions, the data is only partitioned along these two dimensions to form data tiles. The user's current viewport is located at the top left data tile. In Figure 3-2c, only one dimension is being explored (time), which is partitioned into 1D tiles and rendered as a time series visualization. The user's current viewport is located at the center tile along the time dimension.

3.2.2 Interactions Supported by Sculpin

In this thesis, we focus on supporting data exploration through one or more two-dimensional (2D) views, where exploration means that the user can browse, but not modify the underlying dataset. In addition, we assume that users are interacting with the data using consistent, incremental actions that only retrieve a fraction of the underlying dataset. For example, if the user wants to go from zoom level 0 to 4 in Sculpin, she must go through levels 1, 2, and 3 first. Otherwise, users are essentially performing random accesses on the underlying data, which are generally difficult to optimize for any back-end DBMS (*e.g.,* "jumping" to any location in the dataset).

These assumptions define a specific class of exploration interfaces, characterized by the following four rules: (a) the interface supports a finite set of interactions (*i.e.,* no open-ended text boxes); (b) these interactions cannot modify the underlying dataset; (c) each interaction will request only a small fraction of data tiles; and (d) each interaction represents an incremental change to the user's current location in the dataset (*i.e.,* no "jumping"). Note that given rule (c), Sculpin does not currently support interactions that force a full scan of the entire dataset, such as searches (*e.g.,* find all satellite imagery pixels with a snowcover value above 0.7).

3.2.3 Building Data Tiles

To improve performance, Sculpin builds a subset of data tiles in advance (*i.e.,* before the user starts to explore), and stores them on disk in SciDB. First, we consider the comprehensive case, where every tile is computed (or built) in advance, and then describe how Sculpin augments this process to build only a subset of individual tiles. To create visualizations, Sculpin must perform two separate operations on the back-end: a build operation, where Sculpin executes the user's query in the DBMS and stores the results as a separate array; and a fetch operation, where Sculpin retrieves the data to be visualized by issuing a fetch query to the DBMS for the computed array. In this section, we focus on the build operation, where Sculpin builds zoom levels and data tiles in three steps: (1) building a separate materialized view for each zoom level; (2) partitioning each zoom level into non-overlapping blocks of fixed size (*i.e.,* data tiles); and (3) computing any necessary metadata (*e.g.,* data

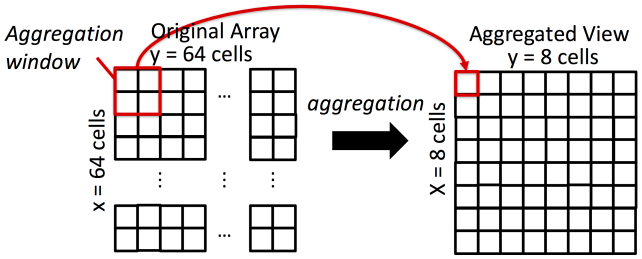


Figure 3-3: A 16x16 array being aggregated down to an 8x8 array with aggregation parameters $(2,2)$. Every 4 cells in the input array (*i.e.,* the red box on the left) becomes a single cell in the aggregated results (*i.e.,* the red box on the right).

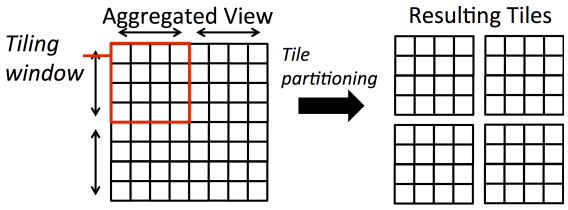


Figure 3-4: A zoom level being partitioned into four tiles, with tiling parameters $(4,4)$.

statistics) for each data tile. The most detailed zoom level (*i.e.,* highest resolution) is just the original query results without any aggregation.

Building Materialized Views: To build a materialized view, we apply an aggregation operation to the user's query, where the aggregation parameters dictate how detailed the resulting zoom level will be. These parameters form a tuple $(j_1,j_2,...,j_d)$, where d is the number of dimensions. Each parameter j specifies an aggregation interval over the corresponding dimension, where every j array cells along this dimension are aggregated into a single cell. Consider Figure 3-3, where we have a 16x16 array (on the left), with two dimensions labeled x and y, respectively. Aggregation parameters of $(2,2)$ correspond to aggregating every 2 cells along dimension x, and every 2 cells along dimension y (*i.e.,* the red box in Figure 3-3). If we compute the average cell value for each (non-overlapping) window in the 16x16 array, the resulting array will have dimensions 8x8 (right side of Figure 3-3).

Partitioning the Views: Next, we partition each computed zoom level into data tiles. To do this, we assign a tiling interval to each dimension, which dictates the number of aggregated cells contained in each tile along this dimension. For example, consider our aggregated 8x8 view in Figure 3-4. If we specify a tiling window of $(4,4)$, Sculpin will partition this view into four separate data tiles, each with the dimensions we specified in

our tiling parameters (4x4).

We choose the aggregation and tiling parameters such that one tile at zoom level i translates to four higher-resolution tiles at level $i+1$. To do this, we calculated our zoom levels bottom-up (*i.e.,* starting at the raw data level), multiplying our aggregation intervals by 2 for each coarser zoom level going upward. We then applied the same tiling intervals to every zoom level. Thus, all tiles have the same dimensions (*i.e.,* tile size), regardless of zoom level.

Note that each zoom level covers the full range of the query result. Therefore, for any tile, we can calculate the corresponding "parent" or "child" tiles that a user might zoom to when interacting with the client-side interface.

Computing Metadata: Last, Sculpin computes any necessary metadata for each data tile. For example, some of our recommendation models rely on data characteristics, or signatures, to be computed for each tile, such as histograms or machine vision features (see Section 4.2 for more detail). As Sculpin processes each tile and zoom level, this metadata is computed and stored in a shared data structure for later use by our prediction engine.

Choosing a Tile Size: Pre-computing tiles ensures that Sculpin provides consistently fast performance across zoom levels. However, choosing a bad tile size can negatively affect performance. For example, increasing the tile size reduces the number of tiles that can be stored in the middleware cache (assuming a fixed cache size), which could reduce Sculpin's prefetching capabilities. In our evaluation (Sections 4.4 and 5.4), we take this into account by varying the number of tiles that are prefetched by Sculpin in our experiments. We plan to perform an in-depth study of how tiling parameters affect performance as future work.

Computing Subsets of Tiles: Instead of computing every zoom level, only a subset of zoom levels are computed in advance, saving the time and space required to compute and store the ignored zoom levels. Specifically, the only the coarsest zoom levels are built in advance. We discuss our materialization optimizations in detail in Chapter 5. Then, once the user starts exploring the data, Sculpin switches its focus to computing (or building) individual data tiles. When Sculpin needs to fetch a tile that has not been built, Sculpin uses the same two-step build and fetch process described above: Sculpin issues one query to

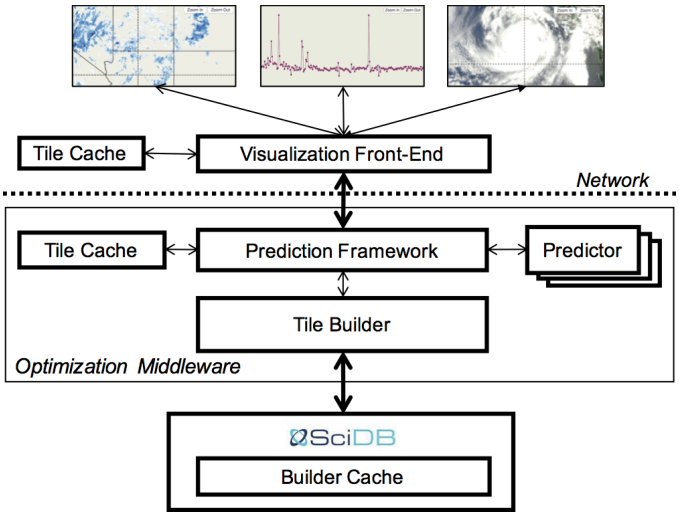


Figure 3-5: A diagram of the Sculpin architecture.

build and store the tile in the DBMS, then another to fetch the built tile from the DBMS. To build a data tile, Sculpin filters the original zoom level query for the exact range matching the tile, then executes this filtered query in the DBMS. We refer to the time spent executing the tile query as the *cost* of the tile. Note that Sculpin treats the DBMS as a black box, and does not assume access to any intermediate query results or any metadata that is not publicly accessible through query API's.

3.3 Architecture

Sculpin contains both client-side and server-side logic to track and predict user interactions, as well as to build, pre-fetch and cache tiles. A diagram of the Sculpin architecture is provided in Figure 3-5[1]. Sculpin has a browser-based front-end on the client which renders visualizations and manages tiles in the browser, fetching tiles from the server when necessary. On the server, Sculpin has a prediction framework for predicting what tiles the user will request in the future, and a tile builder for determining which tiles should be built and when (offline or online). The tile builder issues queries to a back-end DBMS to build tiles. In this section, we describe each major component of the Sculpin architecture.

Client-Side Front-End: The front-end is the sole user-facing component of Sculpin, and is designed to run entirely in a web browser. This component is responsible for creating

[1]Sculpin currently assumes a single-user context. We discuss extending Sculpin to the multi-user case as future work in Chapter 7.

and updating the coordinated visualizations (e.g. see Figure 3-6) that are used to explore data. On the left, Figure 3-6 shows a map view of the western US and Mexico; on the right, a timeline view showing changes in light intensity over time. The user can pan and zoom in any visualization within the coordinated views, which triggers tile requests to the server, and re-rendering of the visualizations (more details are provided in Section 3.4). Sculpin requires knowledge of the user's recent interactions to make predictions. Therefore, this component is also responsible for recording the user's interactions with each visualization of the coordinated views, and for mapping these interactions into tile requests to be sent to the server.

Server-Side Prediction Framework: We developed our prediction framework to support our multidimensional visualization front-end. Using past user interaction input, the framework runs multiple low-level *predictors*, one per visualization in the client-side coordinated view. Each predictor is a dedicated sub-component designed specifically to make predictions for 2D visualizations, and each predictor only has access to past interactions within the associated visualization. The predictors are run in parallel, and space is allocated for the predicted data tiles from each low-level predictor. To efficiently allocate space across predictors, we utilize a multi-level prediction strategy, where we first predict the most likely visualization that the user will interact with next. Extra space is given to the visualization that is deemed more likely to be interacted with next. The final predictions are then consolidated across predictors and sent to the Tile Builder for retrieval.

Server-Side Tile Builder: Tile Builder is responsible for issuing queries to the server-side DBMS to build and fetch tiles, and for managing the disk-based builder cache. The builder cache is finite, so the Tile Builder is also responsible for deciding which tiles to evict when the cache is full. The Tile Builder has two associated processes, an *offline* and an *online* process. The offline process pre-populates the builder cache with the zoom levels that are the most expensive to build. The online process is executed during runtime. In both cases, given a list of candidate tiles from the predictors mentioned above, the Tile Builder performs optimizations to determine which tiles should be built first based on likelihood of use (see Section 5.2 for more detail).

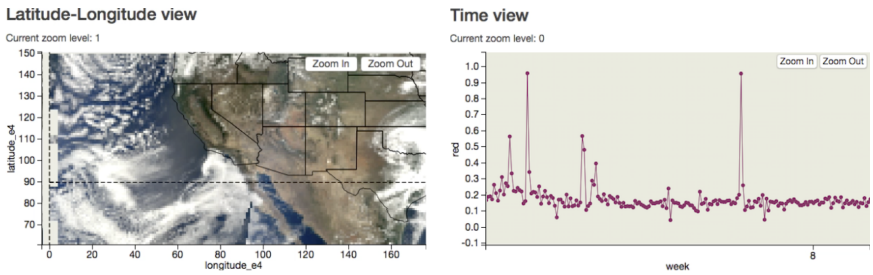


Figure 3-6: A snapshot of a Sculpin coordinated view visualization.

3.4 Front-End Interface Design

In this section, we describe the two major sub-components of the front-end: the Vis Manager, which controls the coordinated view visualization; and the Tile Request Manager, the component responsible for communicating with the server.

3.4.1 Vis Manager

The Vis Manager controls the coordinated view visualizations that the user explores, and tracks the user's interactions. Each visualization renders one or two dimensions from the underlying dataset (*i.e.,* one or two *data dimensions*). Each visualization has a specific configuration that describes the mapping from x and y visualization dimensions (or pixel space) to data attributes or dimensions in the underlying array (data space). The user's current location within each visualization also defines her current position within the dataset. The front end combines the position information from all of the visualizations to compute range filters to find the corresponding data tiles. Therefore, when the user interacts with a visualization, the front-end must fetch a fresh set of tiles corresponding to the new combined data range. The new tiles are then used to re-render all visualizations in the coordinated view.

Button clicks are used to zoom and mouse drags are used to pan in the visualizations. Button clicks shift the visualization by a fixed distance (*e.g.,* zooming in by one level, or panning left by one tile). Mouse drags shift the visualization in the same direction and distance as the corresponding drag action.

The Vis Manager records the start and end coordinates of each interaction in two ways: in pixel space (*i.e.,* where the user started and ended on her computer screen) and data

space (*i.e.,* where the user started and ended within the dataset).

3.4.2 Tile Request Manager

The Tile Request Manager is responsible for managing all requests sent to the server, and all tiles sent to the client. For each interaction, the Tile Request Manager checks a client-side tile cache for the tiles corresponding to this location. If some tiles are missing, the Tile Request Manager issues a request to the server for these tiles, and stores any tiles retrieved from the server in the client-side main memory cache. If the cache is full, tiles are evicted using one of the tile eviction policies described in Section 5.3.1. The Tile Request Manager also periodically sends interaction data to the server to update the Prediction Framework.

3.5 Chapter Summary

In this chapter, we presented the data model and architecture of Sculpin, an exploratory browsing system that supports detail-on-demand (or pan-zoom) browsing of multidimensional datasets. Sculpin adopts a client-server architecture, where the user interacts with a lightweight visualization front-end on a client machine (*e.g.,* a laptop), and the client fetches the data to be visualized from a remote server running a DBMS (here, SciDB). To the best of our knowledge, Sculpin is the first exploratory browsing system to simultaneously reduce interaction latency (*i.e.,* response times), materialization latency, and disk space consumption. To do this, Sculpin contains a server-side middleware layer with several optimization techniques. In Chapters 4 and 5, we discuss the specific optimization techniques utilized in Sculpin to support interactive exploration of massive datasets.

Chapter 4

Dynamic Prefetching of Data Tiles in Sculpin

4.1 Introduction

In the previous chapter, we introduced the data model and general architecture of the Sculpin exploratory browsing system. The purpose of Sculpin is to enable users to quickly and iteratively browse large datasets. The general analysis workflow includes the following steps: 1) the user inputs a complex analysis query to be visualized (*i.e.,* a DBMS query); 2) the visualization tool executes queries in the DBMS to prepare an interactive visualization; and 3) the user interacts with the resulting visualization through panning and zooming interactions. The user can repeat this process with a new DBMS query to explore. In this chapter, we explain the design of the Sculpin prediction framework, which applies novel data pre-fetching techniques to reduce *interaction latencies* in the system, or the time taken by the system to respond to the user's interactions with a rendered visualization (*i.e.,* the user's pans and zooms). However, we note that reducing interaction latencies only addresses one of our performance goals for Sculpin. Specifically, we focus on three major performance goals throughout this thesis:

Goal 1 Reduce interaction latency, so the user can perform panning and zooming interactions quickly.

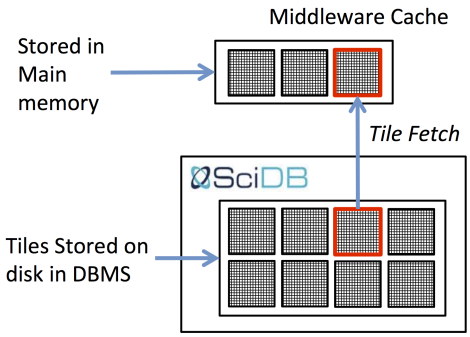


Figure 4-1: A diagram of Sculpin's tile storage scheme.

Goal 2 Reduce materialization latency, so the user can quickly create new visualizations of DBMS queries.

Goal 3 Reduce the disk space consumed by data tiles (*i.e.,* materialized queries), so the user can create visualizations with minimal effort.

We explain how Sculpin supports Goals 2 and 3 in the next chapter.

To reduce interaction latencies in Sculpin (Goal 1), we first consider the full interaction process. Specifically, the user cycles through the following interaction steps when browsing data through a single visualization in Sculpin: (1) she analyzes the result of the previous request, (2) performs an action in the interface to update or refine the request (*e.g.,* zooms in), and then (3) waits for the result to be rendered on the screen. Sculpin eliminates step 3 by prefetching neighboring tiles and storing them in main memory while the user is still in step 1, thereby providing the user with a seamless browsing experience. At the middleware level, we incorporate a main-memory cache for fetching computed tiles, shown in Figure 4-1. When tiles are prefetched, they are copied from SciDB to the cache. However, in a multi-user environment, there may be too little space on the server to cache all neighboring tiles for every user. Furthermore, we may only have time to fetch a small number of tiles before the user's next request. Thus, we must rank the tiles first, and fetch only the most likely candidates.

While prefetching is known to be effective, Sculpin needs access to the user's past interactions with the interface to predict future data requests. We have observed that the client has extensive records of the user's past interactions, which we can leverage to improve our prefetching strategy. For example, the client knows what regions the user has visited in the past, and what actions she has recently performed. One straightforward optimization is to

train a Markov model on the user's past actions, and to use this model to predict the user's future actions [17, 27]. We refer to these prediction techniques as *recommendation models* throughout this thesis.

However, the user's actions are often too complex to be described by a single model (which we will show in Section 4.4). Thus, existing models only cover a fraction of possible analysis goals, leading to longer user wait times due to prediction errors. A comprehensive approach is needed, such that we can consistently prefetch the right tiles over a diverse range of high-level analysis goals.

To address the limitations of existing techniques, we have designed a new two-level predictor for our middleware, which is a sub-component of our prediction framework. This predictor is designed to anticipate the user's future interactions with a single 2D visualization. At the top level, our predictor learns the user's current *analysis phase (*i.e., *her current frame of mind)*, given her most recent actions. The user's analysis phase hints at her current analysis goals, and thus provides context for which actions in the interface she may use to reach her goals. We provide examples of analysis phases in the following paragraph. Furthermore, users frequently employ several low-level browsing patterns within each analysis phase (*e.g.,* panning right three times in a row). Therefore at the bottom level, our predictor runs multiple recommendation models in parallel, each designed to model a specific low-level browsing pattern. Using this two-level design, our predictor tracks changes in the user's current analysis phase, and updates its prediction strategy accordingly. To do this, we increase or decrease the space allotted to each low-level recommendation model for predictions.

Taking inspiration from the Pirolli and Card Sensemaking model [80], we have observed that the space of user interaction patterns can be partitioned into three separate analysis phases: **Foraging** (analyzing individual tiles at a coarse zoom level to form a new hypothesis), **Sensemaking** (comparing neighboring tiles at a detailed zoom level to test the current hypothesis), and **Navigation** (moving between coarse and detailed zoom levels to transition between the previous two phases). The user's goal changes depending on which phase she is currently in. For example, in the Navigation phase, the user is shifting the focus of her analysis from one region in the dataset to another. In contrast, the user's goal

in the Foraging phase is to find new regions that exhibit interesting data patterns.

We consider two separate mechanisms for our low-level recommendation models: (a) learning what to fetch based on the user's past movements (*e.g.,* given that the the user's last three moves were all to "pan right," what should be fetched?) [27]; and (b) using data-derived characteristics, or *signatures*, to identify neighboring tiles that are similar to what the user has requested in the past. We use a Markov chain to model the first mechanism, and a suite of signatures for the second mechanism, ranging from simple statistics (*e.g.,* histograms) to sophisticated machine vision features.

We then take our 2D predictor, which is designed for making predictions for a single 2D visualization, and extend it into a comprehensive multidimensional prediction framework, capable of making predictions in parallel across multiple visualizations. To do this, we run a separate copy of our predictor code, one copy per visualization in the user interface, and allocate space separately to each predictor based on how likely the user is to interact with the corresponding visualization in the interface.

To evaluate our prediction techniques, we conducted a user study, where domain scientists explored satellite imagery data. Our results show that Sculpin achieves (near) interactive speeds for data exploration (*i.e.,* average latency within 500 ms). We also found that Sculpin achieves: (1) dramatic improvements in latency compared with traditional non-prefetching systems (430% improvement in latency); and (2) higher prediction accuracy (25% better accuracy) and significantly lower latency (88% improvement in latency), compared to existing prefetching techniques. We make the following contributions:

1. We propose a new three-phase analysis model to describe how users generally explore array-based data.
2. We present our two-level predictor, with an SVM classifier at the top level to predict the user's current analysis phase, and recommendation models at the bottom to predict low-level interaction patterns.
3. We propose an extension to our original predictor design to make predictions across multiple visualizations, which we developed as a multidimensional prediction framework.
4. We present the results from our user study. Our results show that our prediction ap-

proach provides higher prediction accuracy and significantly lower latency, compared to existing techniques.[1]

4.1.1 Background

Sculpin relies on a diverse set of prediction components and user inputs. Here, we provide an overview of the main concepts utilized in Sculpin.

User Interactions/Moves: The user's interactions with Sculpin are the actions she makes in the front-end interface to explore her data. We also refer to these interactions as *moves*.

User Session: A user session refers to a single session for which the user has logged into Sculpin and explored a single dataset.

Data Model: The Sculpin data model defines: (1) the structure and layout of data tiles, and (2) how to build data tiles. We explain the Sculpin data model in detail in Section 3.2.

Analysis Model: Our analysis model is defined by our three analysis phases, and how these phases interact with each other. We explain our analysis model in detail in Section 4.2.3.

Analysis Phase: The user's current analysis phase represents her frame of mind while exploring data in Sculpin (*i.e.,* Foraging, Navigation, or Sensemaking). Analysis phases can be inferred through the user's interactions; we explain how we predict analysis phases in Section 4.2.3.

Browsing Patterns: Low-level browsing patterns are short chains of interactions repeated by the user (*e.g.,* zooming in three times). We explain how we predict these patterns in Section 4.2.4.

Recommendation Model: A recommendation model is a model used to predict low-level browsing patterns (*e.g.,* Markov chains). Sculpin employs two kinds of recommendation models: Action-Based (Section 4.2.4) and Signature-Based (Section 4.2.4).

[1] Note that we verify Sculpin's performance in the multidimensional context in the following chapter, and focus here on demonstrating our pre-fetching techniques specifically in the 2D case.

4.2 2D Predictor Design

In this section, we describe the methods behind the core component of the prediction framework, the 2D predictor. The goal of the 2D predictor component is to identify changes in the user's browsing patterns when interacting with a single 2D visualization, and to update its prediction strategy accordingly. In this way, our predictor ensures that the most relevant prediction algorithms are being used to prefetch data tiles. To do this, our predictor makes predictions at two separate levels. At the top level, it learns the user's current analysis phase. At the bottom level, it models the observed analysis phase with a suite of recommendation models.

We chose a two-level design because we have found that users frequently switch their browsing patterns over time. In contrast, recommendation models make strict assumptions about the user's browsing patterns, and thus ignore changes in the user's behavior. For example, a Markov chain trained solely on move data (*e.g.,* pan left, zoom out, etc.) relies on the assumption that the user's past moves will always be good indicators of her future actions. However, once the user finds a new region to explore, the panning actions that she used to locate this region will be poor predictors of the future zooming actions she will use to move towards this new region. As a result, we have found that recommendation models only work well in specific cases, making any individual model a poor choice for predicting the user's entire browsing session.

However, if we can learn what a user is trying to do, we can identify the analysis phase that best matches her current goals, and apply the corresponding recommendation model(s) to make predictions. To build the top level of our predictor, we trained a classifier to predict the user's current analysis phase, given her past tile requests. To build the bottom level of our predictor, we developed a suite of recommendation models to capture the different browsing patterns exhibited in our analysis phases. To combine the top and bottom levels, we developed three separate allocation strategies for our middleware cache, one for each analysis phase.

In the rest of this section, we formalize the general prediction problem solved by Sculpin, explain how we map raw interaction data to usable input for existing prediction

techniques, explain the top and bottom level designs for our predictor, and discuss how we combine the two levels using our allocation strategies.

4.2.1 Managing Interaction Data

Though visualizations are generally mapped to two dimensions from the underlying dataset (or two *data dimensions*), users can also explore the data at different zoom levels, which together act as a third interaction "dimension". As such, Sculpin treats zoom levels as a 3D space created from 2D data, where panning occurs along the two data dimensions, and zooming occurs along the third zooming dimension. Though existing systems support this notion of "3D" zooming [63, 60, 17], they fail to support true 3D data exploration (*e.g.,* exploration beyond two data dimensions, such as beyond just latitude and longitude). In contrast, Sculpin supports multidimensional data exploration through a coordinated view visualization design.

To enable certain prediction algorithms, such as Markov models, to generalize beyond specific data tile structures, Sculpin also translates a user's interactions into a small set of directional "moves". Specifically, Sculpin considers ten different prediction "moves": the four cardinal directions (north, south, east, west), the four intercardinal directions (northwest, northeast, southwest, southeast), zoom in, and zoom out. Here, "north" refers to "up", "south" to "down", "west" to "left", and "east" to "right".

The Prediction Framework labels an interaction as a "zoom in" or "zoom out" by comparing the change in zoom level for the interaction. If no change in zoom level has occurred, the Prediction Framework assumes that the user performed a "pan".

4.2.2 Prediction Formalization

Here, we provide definitions for all inputs to and outputs from Sculpin, and a formalization of our general prediction problem.

User Session History: The user's last n moves are constantly recorded by the Tile Request Manager on the frontend and sent to the predictor as an ordered list of user requests: $H = [r_1, r_2, ..., r_n]$. Each request $r_i \in H$ retrieves a particular tile T_{ri}. Note that n (*i.e.,* the

history length) is a system parameter set before the current session starts.

Training Data: Training data is used to prepare the predictor ahead of time, and is supplied as a set of traces: $\{U_1, U_2, ...\}$. Each trace U_j represents a single user session, and consists of an ordered list of user requests: $U_j = [r_1, r_2, r_3, ...]$.

Allocation Strategy: The Tile Builder regularly sends the current allocation strategy to the predictor: $\{k_1, k_2, ...\}$, were k_1 is the amount of space allocated to recommendation model m_1.

General Prediction Problem: Given a user request r for tile T_r, a set of recommender allocations $\{k_1, k_2, ...\}$, and session history H, compute an ordered list of tiles to prefetch $P = [T_1, T_2, T_3, ...]$, where each tile $T_i \in P$ is at most d moves away from T_r. The first tile (T_1) has highest priority when prefetching tiles. d is a system parameter set before the current session starts (default is $d = 1$).

Only a few of these prediction parameters must be specified by the user: allocation strategies for the tile cache (Section 4.2.5); distance threshold d (Section 4.2.4); user history length n (Section 4.2.4); and user traces as training data (Sections 4.2.3-4.2.4).

4.2.3 Top-Level Design

In this section, we explain the three analysis phases that users alternate between while browsing array-based data, and how we use this information to predict the user's current analysis phase.

Learning Analysis Phases

We informally observed several users browsing array-based data in SciDB, searching for common interaction patterns. We used these observed patterns to define a user analysis model, or a general-purpose template for user interactions in Sculpin. Our user analysis model was inspired in part by the well-known Sensemaking model [80]. However, we found that the Sensemaking Model did not accurately represent the behaviors we observed. For example, the Sensemaking model does not explicitly model navigation, which is an important aspect of browsing array data. Thus, we extended existing analysis models to

Table 4.1: Input features for our SVM phase classifier, computed from traces from our user study (see Section 4.4 for more details).

Feature Name	Information Recorded	Accuracy for this Feature
X position (in tiles)	X position	0.676
Y position (in tiles)	Y position	0.692
Zoom level	zoom level ID	0.696
Pan flag	1 (if user panned), or 0	0.580
Zoom-in flag	1 (if zoom in), or 0	0.556
Zoom-out flag	1 (if zoom out), or 0	0.448

match these observed behaviors. We found that our users alternated between three high-level analysis phases, each representing different user goals: Foraging, Sensemaking, and Navigation. Within each phase, users employed a number of low-level interaction patterns to achieve their goals. We model the low-level patterns separately in the bottom half of our predictor, which we describe in detail in Section 4.2.4.

In the Foraging phase, the user is looking for visually interesting patterns in the data and forming hypotheses. The user will tend to stay at coarser zoom levels during this phase, because these levels allow the user to scan large sections of the dataset for visual patterns that she may want to investigate further. In the Sensemaking phase, the user has identified a region of interest (or ROI), and is looking to confirm an initial hypothesis. During this phase, the user stays at more detailed zoom levels, and analyzes neighboring tiles to determine if the pattern in the data supports or refutes her hypothesis. Finally, during the Navigation phase, the user is either zooming out to return to the Foraging phase (*i.e.,* to look for a new ROI to explore), or zooming in to in preparation for the Sensemaking phase (*i.e.,* to analyze a particular ROI).

Predicting the Current Analysis Phase

The top half of our two-level scheme predicts the user's current analysis phase. This problem is defined as follows:

Sub-Problem Definition: given a new user request r and the user's session history H, predict the user's current analysis phase (Foraging, Sensemaking, or Navigation).

To identify the current analysis phase, we apply a Support Vector Machine (SVM)

Algorithm 1 Pseudocode to update the last ROI after each request.

Input: A user request r for tile T_r
Output: ROI, a set of tiles representing the user's last visited ROI.

```
1:  ROI ← {}
2:  temp_ROI ← {}
3:  inFlag ← False
4:  procedure UPDATEROI(r)
5:      if r.move ="zoom-in" then
6:          inFlag ← True
7:          temp_ROI ← {T_r}
8:      else if r.move ="zoom-out" then
9:          if inFlag = True then
10:             ROI ← temp_ROI
11:         inFlag ← False
12:         temp_ROI ← {}
13:     else if inFlag = True then
14:         add T_r to temp_ROI
15:     return ROI
```

classifier, similar to the work by Brown *et al.* [14]. SVM's are a group of supervised learning techniques that are frequently used for classification and regression tasks. We used a multi-class SVM classifier with a radial basis function (or RBF) kernel. We implemented our classifier using the LibSVM Java Library[2].

To construct an input to our SVM classifier, we compute a feature vector using the current request r, and the user's previous request $r_n \in H$. The format and significance of each extracted feature in our feature vector is provided in Table 4.1. Because this SVM classifier only learns from interaction data and relative tile positions, we can apply our classification techniques to any dataset that is amenable to a tile-based format. To build a training dataset for the classifier, we collected user traces from the 18 participants of our user study. We then hand-labeled each user request from the user traces with its corresponding analysis phase. We describe our user study and evaluate the accuracy of the analysis phase classifier in Section 4.4.

[2]https://github.com/cjlin1/libsvm

4.2.4 Bottom-Level Design

Once the user's current analysis phase has been identified, Sculpin employs the corresponding recommendation model(s) to predict specific tiles. Sculpin runs these models in parallel, where each model is designed to predict specific low-level browsing patterns. The space allocated to each model (*i.e.,* number of tiles that each model can prefetch) is adjusted by Sculpin over time, based on how effective the model is for the current analysis phase. These recommendation models can be categorized into two types of predictions: (a) *Action-Based (AB)*: learning what to predict from the user's previous moves (*e.g.,* pans and zooms); and (b) *Signature-Based (SB)*: learning what to predict by using data characteristics, or *signatures*, from the tiles that the user has recently requested (*e.g.,* histograms). For an individual recommendation model m, the prediction problem is as follows:

Sub-Problem Definition: given a user request r, a set of candidate tiles for prediction C, and the session history H, compute an ordering for the candidate tiles $P_m = [T_1, T_2, ...]$. The ordering signifies m's prediction of how relatively likely the user will request each tile in C. The predictor trims P_m as necessary, depending on the amount of space allocated to m.

Here, we describe: (1) the inputs and outputs for our recommendation models; (2) how the individual models were implemented; and (3) how we allocate space to each model per analysis phase.

General Recommendation Model Design

Our Signature-Based recommendation model requires one additional input in order to make predictions: the last location in the dataset that the user explored in detail, which we refer to as the user's most recent *Region of Interest*, or ROI. Here we explain how we derive C, and the user's most recent ROI.

Candidate Tiles for Prediction: We compile the set of candidate tiles by finding all tiles that are at most d moves away from r. For example, $d = 1$ represents all tiles that are exactly one move away from r.

Algorithm 2 Pseudocode showing the Markov chain transition frequencies building process.

Input: For PROCESSTRACES, a set of user traces, and sequence length n.
Output: F, computed transition frequencies.

```
 1: procedure PROCESSTRACES({U_1, U_2, ..., U_j, ...}, n)
 2:     F ← {}
 3:     for user trace U_j do
 4:         V_j ← GETMOVESEQUENCE(U_j)
 5:         F ← UPDATEFREQUENCIES(V_j, F, n)
 6:     return F
 7: procedure GETMOVESEQUENCE(U_j)
 8:     V_j ← [ ]
 9:     for i = 1, 2, ..., where i ≤ | V_j | do
10:         V_j[i] ← U_j[i].move
11:     return V_j
12: procedure UPDATEFREQUENCIES(V_j = [v_1, v_2, v_3, ...], F, n)
13:     for i = n+1, n+2, ..., where n < i ≤ | V_j | do
14:         F[sequence(v_{i-n}, v_{i-(n-1)}, v_{i-(n-2)}, ..., v_{i-1}) → v_i] += 1
15:     return F
```

Most Recent ROI: We represent the user's most recent ROI as a set of data tiles, and use a simple heuristic to compute this set; the pseudocode is provided in Algorithm 1. When a new user request is received, the predictor calls UPDATEROI to update the user's most recent ROI. To find the most recent ROI, this heuristic searches through H for a match to the following pattern: one *zoom-in*, followed by zero or more *pan*'s, followed by one *zoom-out*. In lines 5-7 of Algorithm 1, a *zoom-in* triggers the collection of a new temporary ROI ($temp_{ROI}$), and the requested tile T_r is added to $temp_{ROI}$ (line 7). We track *zoom-in*'s using the $inFlag$ variable (line 6). In contrast, an observed *zoom-out* tells the predictor to stop adding tiles to $temp_{ROI}$ (lines 8-12). If the $inFlag$ was set while the zoom-out occurred, we replace the user's old ROI with $temp_{ROI}$ (lines 9-10). Then, $temp_{ROI}$ is reset (line 12). Last, if $r.move = pan$ while the $inFlag$ is true, T_r (*i.e.,* the requested tile) is added to $temp_{ROI}$ (lines 13-14).

Actions-Based (AB) Recommender

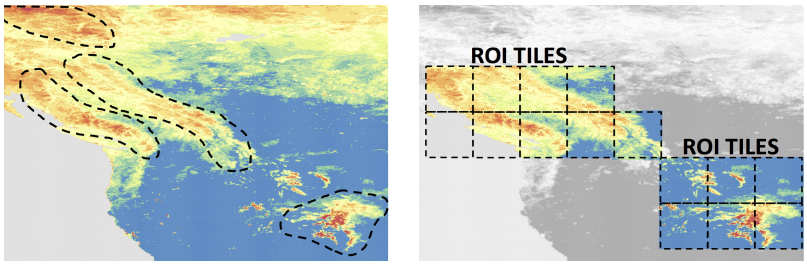
As the user moves to or from ROI's, she is likely to consistently zoom or pan in a predictable way (*e.g.,* zoom out three times). Doshi *et al.* leverage this assumption in their Mo-

mentum model, which predicts that the user's next move will match her previous move [27]. We expand on this idea with our AB recommender, which builds an n-th order Markov chain from users' past actions.

To build the Markov chain, we create a state for each possible sequence of moves, where we only consider sequences of length n (*i.e.,* the length of H). For example, if $n = 3$, then the following are two sequences that would have their own states in the Markov chain: panning left three times (*i.e.,* $left$, $left$, $left$), and zooming out twice and then panning right (*i.e.,* *out*, *out*, *right*). After creating our states, we create an outgoing transition from each state for every possible move the user can make in the interface. In the $n = 3$ case, if the user is in state ($left$, $left$, $left$) and then decides to pan right, we represent this as the user taking the edge labeled "*right*" from the state ($left$, $left$, $left$) to the state ($left$, $left$, *right*).

We learn transition probabilities for our Markov chains using traces from our user study; the traces are described in Section 4.2.2. Algorithm 2 shows how we calculate the transition frequencies needed to compute the final probabilities. For each user trace U_j from the study, we extract the sequence of moves observed in the trace (lines 7-11). We then iterate over every sub-sequence of length n (*i.e.,* every time a state was visited in the trace), and count how often each transition was taken (lines 12-15). To do this, for each sub-sequence observed (*i.e.,* for each state observed from our Markov chain), we identified the move that was made immediately after this sub-sequence occurred, and incremented the relevant counter (line 14).

Note that with this design, our Markov chains could have thousands of states. Thus many of these transition probabilities may not be calculable directly from the training data. To address this issue, we utilize well-known techniques from natural language processing that were developed to address a similar problem: calculating probabilities for unobserved word sequences. To fill in missing counts, we apply Kneser-Ney smoothing, a well-studied smoothing method in natural language processing for Markov chains [20]. We used the BerkeleyLM [78] Java library to implement our Markov chains.



(a) **Potential snow cover ROI's in the US and Canada.** (b) **Tiles in the user's history, after visiting ROI's from (a).**

Figure 4-2: Example ROI's in the US and Canada for snow cover data. Snow is orange to yellow, snow-free areas in green to blue. Note that (a) and (b) span the same latitude-longitude range.

Table 4.2: Features computed over individual array attributes in Sculpin to compare data tiles for visual similarity.

Signature	Measures Compared	Visual Characteristics Captured
Normal Distribution	Mean, standard deviation	average position/color/size of rendered datapoints
1-D histogram	histogram bins	position/color/size distribu--tion of rendered datapoints
SIFT	histogram built from clustered SIFT descriptors	distinct "landmarks" in the visualization (*e.g.,* clusters of orange pixels)
DenseSIFT	same as SIFT	distinct "landmarks" *and* their positions in the visualization

Signature-Based (SB) Recommender

The goal of our SB recommender is to identify neighboring tiles that are visually similar to what the user has requested in the past. For example, in the Foraging phase, the user is using a coarse view of the data to find new ROI's to explore. When the user finds a new ROI, she zooms into this area until she reaches her desired zoom level. Each tile along her zooming path will share the same visual features, which the user depends on to navigate to her destination. In the Sensemaking phase, the user is analyzing visually similar data tiles at the same zoom level. One such example is when the user is exploring satellite imagery of the earth, and panning to tiles within the same mountain range.

Consider Figure 4-2a, where the user is exploring snow cover data derived from a satel-

lite imagery dataset. Snow is colored orange, and regions without snow are blue. Thus, the user will search for ROI's that contain large clusters of orange pixels, which are circled in Figure 4-2a. These ROI's correspond to mountain ranges.

Given the user's last ROI (*i.e.,* the last mountain range the user visited), we can look for neighboring tiles that look similar (*i.e.,* find more mountains). Figure 4-2b is an example of some tiles that may be in the user's history if she has recently explored some of these ROI's, which we can use for reference to find new ROI's.

We measure visual similarity by computing a diverse set of tile *signatures*. A signature is a compact, numerical representation of a data tile, and is stored as a vector of double-precision values. Table 4.2 lists the four signatures we compute in Sculpin. All of our signatures are calculated over a single SciDB array attribute. The first signature in Table 4.2 calculates the average and standard deviation of all values stored within a single data tile. The second signature builds a histogram over these array values, using a fixed number of bins.

We also tested two machine vision techniques as signatures: the scale-invariant feature transform (SIFT), and a variant called denseSIFT (signatures 3 and 4 in Table 4.2). SIFT is used to identify and compare visual "landmarks" in an image, called *keypoints*. Much like how seeing the Statue of Liberty can help people distinguish pictures of New York city from pictures of other cities, SIFT keypoints help Sculpin compare visual landmarks in two different visualizations (*e.g.,* two satellite imagery heatmaps with similar clusters of orange pixels, or two line charts showing unusually high peaks in heart-rate signals). We used the OpenCV library to compute our SIFT and denseSIFT signatures[3].

Algorithm 3 outlines how we compare candidate tiles to the user's last ROI using these signatures. We first retrieve all four signatures for each candidate tile (lines 3-4). We also retrieve these four signatures for each ROI tile on lines 5-6. We explain how we identify ROI tiles in Section 4.2.4. Then we compute how much each candidate tile (T_A) deviates from each ROI tile (T_B), with respect to each signature (lines 7-8). To do this, a distance function for the signature is applied to the candidate tile and ROI tile (denoted as $dist_{S_i}$ in Algorithm 3). Since our signatures do not automatically account for the physical distance

[3]http://opencv.org

between T_A and T_B, we apply a penalty to our signature distances based on the Manhattan distance between the tiles. Since all four of our current signatures produce histograms as output, we use the Chi-Squared distance metric as the distance function for all signatures. We then normalize the computed distance values (lines 10-11).

To produce a single distance measure for a given candidate-ROI pair, we treat the four resulting distance measures as a single vector, and compute the ℓ^2-norm of the vector (lines 12-13). To adjust how much influence each signature has on our final distance measurements, we can modify the ℓ^2-norm function to include weights for each signature. All signatures are assigned equal weight by default, but the user can update these weight parameters as necessary.

$$\ell^2_{weighted}(A,B) = \sqrt{\sum_{S_i} w_i (d_{i,A,B})^2}$$

At this point, there will be multiple distance values calculated for each candidate tile, one per ROI tile. For example, if we have four ROI tiles, then there will be four distance values calculated per candidate tile. We sum these ROI tile distances, so we have a single distance value to compare for each candidate tile (lines 14-15). We then rank the candidates by these final distance values.

Note that it is straightforward to add new signatures to the SB recommender. To add a new signature, one only needs to add: (1) an algorithm for computing the signature over a single data tile, and (2) a new distance function for comparing this signature (if the Chi-Squared distance is not applicable).

4.2.5 Cache Allocation Strategies

In this section, we describe the recommendation models associated with each analysis phase, and how we use this information to allocate space to each recommender in our tile cache.

In the Navigation phase, the user is zooming and panning in order to transition between the Foraging and Sensemaking phases. Thus, we expect the AB recommendation model to be most effective for predicting tiles for this phase, and allocate all available cache space to this model.

Algorithm 3 Computes the visual distance of each candidate tile, with respect to a given ROI.

Input: Signatures S_1-S_4, candidate tiles, ROI tiles
Output: A set of distance values D

for Signature S_i, $i = 1-4$ **do**
    $d_{i,MAX} \leftarrow 1$
    for each candidate tile T_A **do**
        Retrieve signature $S_i(T_A)$
    for each ROI tile T_B **do**
        Retrieve signature $S_i(T_B)$
    for each candidate/ROI pair (T_A,T_B) **do**
        $d_{i,A,B} \leftarrow 2^{d_{manh}(T_A,T_B)-1}[dist_{S_i}(S_i(T_A),S_i(T_B))]$
        $d_{i,MAX} \leftarrow max(d_{i,MAX},d_{i,A,B})$
    for each candidate/ROI pair (T_A,T_B) **do**
        $d_{i,A,B} \leftarrow \frac{d_{i,A,B}}{d_{i,MAX}}$
for each candidate/ROI pair (T_A,T_B) **do**
    $d_{A,B} \leftarrow \frac{\sqrt{\sum_{S_i} w_i (d_{i,A,B})^2}}{d_{physical}(A,B)}$
for each candidate tile T_A **do**
    $d_A \leftarrow \sum_B d_{A,B}$
return $D = \{d_1, d_2, ..., d_A, ...\}$

In the Sensemaking phase, the user is mainly panning to neighboring tiles with similar visual features. Therefore, we expect the SB recommendation model to perform well when predicting tiles for this phase, and allocate all available cache space to this model.

In the Foraging phase, the user is using visual features as cues for where she should zoom in next. When the user finds a ROI that she wants to analyze, the tiles she zooms into to reach this ROI will share the same visual properties. Thus, the SB model should prove useful for this phase. However, the user will also zoom out several times in a row in order to return to the Foraging phase, exhibiting a predictable pattern that can be utilized by the AB model. Therefore, we allocate equal amounts of space to both models for this phase.

4.3 Prediction Framework Design

Unfortunately, we have found that the Sculpin predictor becomes unnecessarily complicated when extended to the multidimensional prediction case. For example, the Sculpin SB recommender ranks the tiles that surround the user's current location to make predic-

tions. However, when the user has access to several different dimensions at any given time, this results in $\prod_d t_d$ tiles that can be predicted, where d is a navigable dimension, and t_d is the number of tiles that are one interaction away from the user's current location along this dimension. This number could be exponential in the number of dimensions, making a ranking-based prediction approach prohibitively expensive to apply for multidimensional datasets.

In response, we propose a new framework for supporting multi-dimensional prediction. The key idea is to constrain the multi-dimensional prediction problem by exploiting the core design principles driving *coordinated view visualizations* [86], a very common and well-studied interface design for multidimensional data exploration. Specifically, Sculpin takes into account that the user can only explore one visualization at a time in a standard coordinated view design. By extension, the user can only interact with a limited number of data dimensions at any given time, since each visualization within the Sculpin interface is associated with a small number of dimensions from the underlying dataset. For example, map visualizations are generally associated with two data dimensions (latitude and longitude). Note that we focus on *data* dimensions; we manage *interaction* dimensions (*e.g.,* zooming) separately (see Section 4.2.1 for more details).

Using these insights, we can constrain our dimensionality problem by re-framing it as a multi-level prediction problem: before we predict which interactions the user will perform, we first predict which visualization these interactions will be performed on. If we can accurately predict which visualization the user will interact with next, we are left with a familiar prediction sub-problem: identifying which interactions the user will perform within the given visualization. Once the next visualization is identified, we can use existing prediction algorithms (*i.e.,* or predictors) to pre-fetch relevant tiles.

In this section, we describe: 1) our multi-level approach to tile prediction across multiple dimensions, and 2) our new strategy for allocating pre-fetching space across multiple visualizations.

4.3.1 Formalization

Here, we describe the inputs to the Sculpin Prediction Framework, and define the multidimensional prediction problem.

Inputs for Individual Predictors: The predictors only require access to the user's recent request history H. We define a user request $r \in H$ as a single message sent from the client Tile Request Manager to the server to fetch new tiles after a user interaction. r contains the following information: the start and end coordinates of the user's last interaction (as defined in Section 3.4.1), and a list of tile ID's to be fetched. A user session can be defined as an ordered list of user requests $U = [r_1, r_2, ...]$.

Sculpin Inputs: The input to the Framework is simply a list of interaction histories $[H_1, H_2, ...]$, one history for each of the coordinated visualizations. For a given visualization v, $H_v = \{r : r \text{ was triggered by } v\}$. We also assume that Sculpin has access to a set of past user sessions to train the Prediction Framework offline.

Prediction Problem: Given the user's recent interaction history H and the user's last request ($r_n \in H$), the prediction framework returns an ordered list of tile candidates $P = [T_1, T_2, ...]$, representing the most likely tiles the user will request in the future. The number of candidates is equal to the total space allocated for pre-fetching tiles in Sculpin.

4.3.2 Multidimensional Data Tile Prediction

Here, we explain how we combine the Vis Selector (top level) and the 2D predictors (bottom level) to form a multi-level prediction design for pre-fetching multidimensional data tiles.

Vis Selector

To enable the user to switch between data dimensions (*e.g.,* from latitude-longitude to time), all possible dimensions have to be on the screen, so the user can initiate an operation on any one of them. This takes the form of interacting with one of the visualizations in the coordinated view (*e.g.,* map view versus time view). When an interaction occurs, this signals a clear switch in dimensions, enabling Sculpin to learn the current context, and

make predictions accordingly.

Given the user's recent interaction history, Sculpin first needs to predict which visualization the user will interact with next. With knowledge of the user's next visualization, Sculpin can focus on pre-fetching tiles that are reachable through interactions with this specific visualization. The specific sub-problem solved by the Vis selector is: given the user's recent interaction history H, and a list of visualizations V = [v_1,v_2,...], select the visualization that the user is most likely to interact with next.

To better understand the visualization selection problem, we informally observed users exploring satellite imagery using a prototype interface similar to the Sculpin front-end. We found that users tend to interact with the same visualization several times, before switching to a different one. We selected a momentum-based approach to predicting the user's next visualization [27], where the user's previous visualization is likely to be the next visualization that the user will interact with next.

Running Multiple Predictors

Given the user's next visualization v and request histories for each visualization [H_1,H_2,...], the Prediction Framework returns a list of tiles $P = [T_1, T_2, ...]$, representing the tiles that the user will request in the future.

To capture the interaction patterns exhibited within individual visualizations, Sculpin runs multiple 2D predictors in parallel. Each Predictor uses its own separate suite of low-level recommendation models to predict which tiles will be requested for the associated visualization. The recommendation models are trained on past user sessions, where each user session is filtered for the associated visualization: $U_v = \{r : r \text{ was triggered by } v\}$.

Each predictor produces an ordered list of candidate tiles to be pre-fetched, where the most likely tile to be requested is first in the list. Each list is truncated based on the amount of pre-fetching space allocated to the associated predictor. Then, the lists are merged in a round-robin configuration to create a single master list (duplicates are skipped). Using round-robin means that Sculpin assumes equal weighting across predictors. One could apply weights to each predictor, and use a more sophisticated merging strategy.

The Prediction Framework then checks the main memory tile cache for the predicted

tiles. All un-cached predictions from the master list are then sent to the Tile Builder for retrieval. Once the tiles are retrieved, they are batch-inserted into the server-side main memory tile cache.

Pre-fetching space is allocated separately to each predictor, where the predictor associated with the most likely visualization receives the majority of the space in the cache for predicting data tiles. The remaining space is divided equally among the remaining predictors. Sculpin gives the most promising predictor three quarters of the pre-fetching space by default.

4.4 Experiments: Evaluating the 2D Predictor

The 2D predictor is the cornerstone of Sculpin: the 2D predictor powers the prediction framework, which in turn supports the other optimizations in Sculpin. Thus, before we can gauge the overall effectiveness of the Sculpin system, we must first evaluate the 2D predictor. In this section, we explain how we evaluated the 2D predictor through a user study with scientists exploring NASA MODIS satellite imagery, and the results of this study.

Although the goal behind our prediction techniques is to reduce interaction latency, we will demonstrate in Section 4.4.5 that there is a linear (constant factor) correlation between latency and the accuracy of the prediction algorithm. As such, we claim that we can improve the observed interaction latency in Sculpin by reducing the number of prediction errors that occur when prefetching tiles ahead of the user. Our aim in this section is to show that Sculpin provides significantly better prediction accuracy, and thus lower interaction latency, when compared to existing prefetching techniques.

We validate our claims about user exploration behavior through a user study on NASA MODIS satellite imagery data, and evaluate the prediction accuracy of our 2D predictor using traces collected from the study. To validate our hypothesis that prediction accuracy dictates the overall latency of the system, we also measured the average latency observed in Sculpin for each of our prediction techniques.

To test the accuracy of our predictor, we conducted three sets of evaluations. We first

evaluate each prediction level separately. At the top level, we measure how accurately we can predict the user's current analysis phase. At the bottom level, we measure the overall prediction accuracy of each recommendation model, with respect to each analysis phase, and compare our individual models to existing techniques. Then we compare the accuracy of the full predictor to our best performing individual recommendation models, as well as existing techniques. Last, we evaluate the relationship between accuracy and latency, and compare the overall latency of our full predictor to existing techniques.

4.4.1 MODIS Dataset

The NASA MODIS is a satellite instrument that records imagery data. This data is originally recorded by NASA in a three-dimensional array (latitude, longitude and time). Each array cell contains a vector of wavelength measurements, where each wavelength measurement is called a MODIS "band."

One use case for MODIS data is to estimate snow depths in the mountains. One well-known MODIS snow cover algorithm, which we apply in our experiments, is the Normalized Difference Snow Index (NDSI) [85]. The NDSI indicates whether there is snow at a given MODIS pixel (*i.e.,* array cell). A high NDSI value (close to 1.0) means that there is snow at the given pixel, and a low value (close to -1.0) corresponds to no snow cover. The NDSI uses two separate wavelength measurements (*i.e.,* MODIS bands) to calculate this. We label the two bands used in the NDSI as μ_{VIS} for visible light, and μ_{SWIR} for short-wave infrared. The NDSI is calculated by applying the following function to each cell of the MODIS array, which calculates the normalized difference between the corresponding MODIS bands within the array cell:

$$NDSI = \frac{(\mu_{VIS} - \mu_{SWIR})}{(\mu_{VIS} + \mu_{SWIR})}.$$

It is straightforward to translate this transformation into a user-defined function (UDF) in SciDB.

Modifications for User Study

Our test dataset consisted of NDSI measurements computed over one week of raw NASA MODIS data, where the temporal range of the data was from late October to early November of 2011. We downloaded the raw data directly from the NASA MODIS website[4], and used SciDB's MODIS data loading tool to load the data into SciDB. We applied the NDSI to the raw MODIS data as a user-defined function, and stored the resulting NDSI calculations in a separate array. The NDSI array was roughly 10TB in size when stored in SciDB.

Given that our 2D predictor is designed specifically to work with a single 2D visualization, we modified the MODIS dataset to match this data structure. Prior to the study, The NDSI dataset was aggregated into a single, one-week time window, reducing the total dimensions from three (latitude, longitude, time) to two (latitude and longitude only). This enabled us to visualize the MODIS data using a single latitude-longitude map. The NDSI dataset contained four numeric attributes: maximum, minimum and average NDSI values; and a land/sea mask value that was used to filter for land or ocean pixels in the dataset. Sculpin's tile computation process resulted in nine total zoom levels for this dataset, where each level was a separate layer of data tiles.

Calculating the NSDI in SciDB

In this section, we explain how to compute the NSDI in SciDB. We assume that we already have a NDSI UDF written in SciDB, which we refer to as "ndsi_func".

Let S_{VIS} and S_{SWIR} be the SciDB arrays containing recorded data for their respective MODIS bands. We use two separate arrays, as this is the current schema supported by the MODIS data loader for SciDB [81]. S_{VIS} and S_{SWIR} share the same array schema. An example of this schema is provided below.

$$S_{VIS/SWIR}(\text{reflectance})[latitude, longitude].$$

The array attributes are denoted in parentheses (reflectance) and the dimensions are shown in brackets (latitude and longitude). The attributes represent the MODIS band measurements recorded for each latitude-longitude coordinate.

[4] http://modis.gsfc.nasa.gov/data/

The following is the SciDB query we execute to compute the NDSI over the S_{VIS} and S_{SWIR} arrays:

Query 4.1: SciDB query to apply the NDSI.

```
1 store(
2     apply(
3         join(S_VIS, S_SWIR),
4         ndsi,
5         ndsi_func(S_VIS.reflectance,
6             S_SWIR.reflectance)
7     ),
8     NDSI
9 );
```

We first perform an equi-join, matching the latitude-longitude coordinates of the two arrays (line 3). Note that SciDB implicitly joins on dimensions, so latitude and longitude are not specified in the query. We then apply the NDSI to each pair of joined array cells by calling the "ndsi_func" UDF (lines 5-6). We pass the reflectance attribute of S_{VIS} and the reflectance attribute of S_{SWIR} to the UDF. We store the result of this query as a separate array in SciDB named *NDSI* (line 8), and the NDSI calculations are recorded in a new "ndsi" attribute in this array (line 4).

4.4.2 Experimental Setup

Hardware/Software setup

The Sculpin front-end for the study was a web-based visualizer. The D3.js Javascript library was used to render data tiles. We describe the interface in more detail below.

The data was partitioned across two servers running SciDB version 13.3. Each server had 24 cores, 47GB of memory, and 10.1 TB of disk space. Both servers ran Ubuntu Server 12.04. The first server was also responsible for running the Sculpin middleware (prediction engine and cache manager), which received tile requests from the client-side interface.

Note that in these experiments, only a single Sculpin predictor was needed, since par-

ticipants only interacted with a single 2D map visualization.

Measuring Accuracy

The number of cache misses (*i.e.,* prediction accuracy) directly impacts whether delays occur in Sculpin, and thus also determines the length of user wait times (*i.e.,* the interaction latency). Therefore, we used prediction accuracy as one of our primary metrics for comparison, similar to Lee *et al.* [56]. To compute this, we ran our models in parallel while stepping through tile request logs, one request at a time. For each requested tile, we collected a ranked list of predictions from each of our recommendation models, and recorded whether the next tile to be requested was located within the list.

We simulated space allocations in our middleware cache by varying k in our accuracy measurements. Thus measuring prediction accuracy becomes equivalent to measuring the hit rate of our tile cache. For example, $k = 2$ meant that Sculpin only had space to fetch two tiles before the user's next request. We varied k from 1 to 8 in our experiments. At $k = 9$, we are guaranteed to prefetch the correct tile, because the interface only supports nine different moves: zoom out, pan (left, right, up, down), and zoom in (users could zoom into one of four tiles at the zoom level below).

Given that Sculpin prefetches new tiles after every request, we found that having Sculpin predict further than one move ahead did not actually improve accuracy. Therefore, predicting beyond the user's next move was irrelevant to the goals of these experiments, and we only considered the tiles that were exactly one step ahead of the user. We leave prefetching more than one step ahead of the user as future work.

Comparing with Existing Techniques

To compare our two-level prediction engine with existing techniques, we implemented two models proposed in [27], the "Momentum" and "Hotspot" models. Several more recent systems, such as ATLAS [17] and ImMens [63] apply very similar techniques (see Chapter 6 for more information).

Momentum: The Momentum model assumes that the user's next move will be the same as her previous move. To implement this, the tile matching the user's previous move

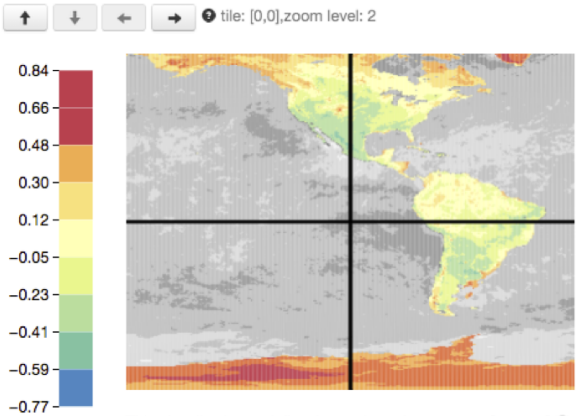


Figure 4-3: User study browsing interface.

is assigned a probability of 0.9, and the eight other candidates are assigned a probability of 0.0125. Note that this is a Markov chain, since probabilities are assigned to future moves based on the user's previous move.

Hotspot: The Hotspot model is an extension of the Momentum model that adds awareness of popular tiles, or *hotspots*, in the dataset. To find hotspots in the NDSI dataset, we counted the number of requests made for each tile visited in our user study, and chose the tiles with the most requests. When the user is not close to any hotspots, the Hotspot model defaults to the behavior of the Momentum model. When a hotspot is nearby, the Hotspot model assigns a higher ranking to any tiles that bring the user closer to that hotspot, and a lower ranking to the remaining tiles. We trained the Hotspot model on trace data ahead of time. This training process took less than one second to complete.

4.4.3 User Study

To ascertain whether prefetching was a viable strategy for exploring multidimensional scientific datasets, we worked directly with earth and ocean scientists at the University of California Santa Barbara (UCSB) and the University of Washington (UW) to: (1) choose a use case of interest to our collaborators (MODIS snow cover); and (2) develop a set of search tasks for this use case that domain scientists with diverse backgrounds and skill sets could complete during the study. In this section, we outline the study design, and validate whether our analysis phases are an appropriate classification of user behavior using results from the study. We avoided biasing the behavior of our study participants by caching all

data tiles in main memory while the study was being conducted. This prevented our participants from choosing their movements based on response time (*e.g.,* avoiding zooming out if it is slower than other movements). This also ensured that all study participants had the same browsing experience throughout the study.

Participants

The study consisted of 18 domain scientists (graduate students, post doctoral researchers, and faculty). Most of our participants were either interested in or actively working with MODIS data. Participants were recruited at UW and UCSB.

Study Procedure

Each participant read and signed a consent form prior to participating in the study. Participants completed the study on their own devices (either a desktop or laptop computer), in their own work offices. Every participant was observed in-person as they completed the study. We provided each participant with instructions on how to use our visualization tool at the beginning of the study. Participants were then given the opportunity to explore the interface of our tool for five minutes, and were encouraged to ask questions about the instructions and the interface. After participants completed the study, they filled out a five-minute debrief survey about their experience with the tool.

Browsing Interface

Figure 4-3 is an example of the client-side interface. The interface used for the study was an early Sculpin prototype that used buttons for panning instead of mouse drags. Each visualization in the interface represented exactly one data tile. Participants (*i.e.,* users) used directional buttons (top of Figure 4-3) to move up, down, left, or right. Moving up or down corresponded to moving along the latitude dimension in the NDSI dataset, and left or right to the longitude dimension. Each directional move resulted in the user moving to a completely separate data tile. User's left clicked on a quadrant to zoom into the corresponding tile, and right clicked anywhere on the visualization to zoom out.

Directional buttons ensured that users' actions mapped to specific data tiles. Though different from existing geospatial interfaces (*e.g.,* Google Maps), our browsing interface provides clear and efficient button-based navigation through the dataset. Furthermore, study participants commented that this navigation design is useful when selecting specific data ranges for further analysis.

Browsing Tasks

Participants completed the same search task over three different regions in the NDSI dataset. For each region, participants were asked to identify four data tiles (*i.e.,* four different visualizations) that met specific visual requirements. The tasks were as follows:

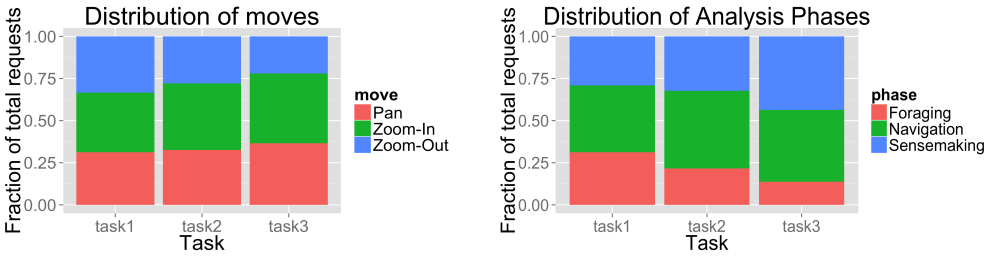
1. Find four data tiles in the continental United States at zoom level 6 with the highest NDSI values.
2. Find four data tiles within western Europe at zoom level 8 with NDSI values of .5 or greater.
3. Find 4 data tiles in South America at zoom level 6 that contain NDSI values greater than .25.

A separate request log was recorded for each user and task. Therefore, by the end of the study we had 54 user traces, each consisting of sequential tile requests.

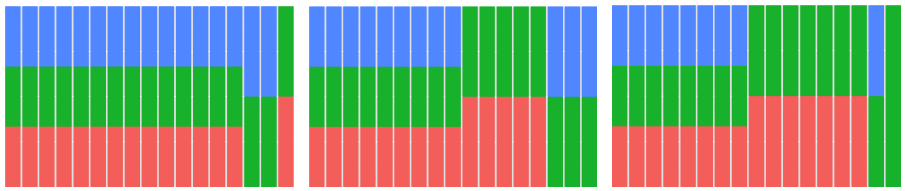
Post-Study: General Observations

The most popular ROI's for each task were: the Rocky Mountains for Task 1, Swiss Alps for Task 2, and Andes Mountains for Task 3. The average number of requests per task are as follows: 35 tiles for Task 1, 25 tiles for Task2, and 17 tiles for Task 3. The mountain ranges in Tasks 2 and 3 (Europe and South America) were closer together and had less snow than those in task 1 (US and Southern Canada). Thus, users spent less time on these tasks, shown by the decrease in total requests.

We also tracked whether the request was a zoom in, zoom out, or pan. Figure 4-4a shows the distribution of directions across all study participants, recorded separately for each task. We see that for all tasks, our study participants spent the most time zooming



(a) **Disribution of moves** (b) **Distribution of Phases**



(c) **Distribution of Moves (Task1)** (d) **Distribution of Moves (Task 2)** (e) **Distribution of Moves (Task3)**

Figure 4-4: Distribution of moves (a) and phases (b), averaged across users, partitioned by task; distribution of moves computed for each user for Task 1 (c), Task 2 (d), and Task 3 (e); each user's distribution of moves is represented as a single column. In Figures (c), (d), and (e): panning is red, zooming in is green, and zooming out is blue; users with similar move distributions are grouped together.

in. This is because users had to zoom to a specific zoom level for each task, and did not have to zoom back out to the top level to complete the task. In tasks 1 and 2, users panned and zoomed out roughly equally. In task 3, we found that users clearly favored panning more than zooming out. We also found that large groups of users shared similar browsing patterns, shown in Figures 4-4c-4-4e. For example in Task 1, we observed that 14 participants panned, zoomed in, and zoomed out roughly equally throughout the task, represented by the first 14 columns of Figure 4-4c. These groupings further reinforce the reasoning behind our analysis phases, showing that most users can be categorized by a small number of specific patterns within each task, and even across tasks.

Evaluating Our Three Analysis Phases

To demonstrate some of the patterns that we found in our user traces, consider Figure 4-5, which plots the change in zoom level over time for one of our user traces. The coarsest zoom level is plotted at the top of Figure 4-5, and the most-detailed zoom level was plotted at the bottom. The x-axis represents each successive tile request made by this user. A

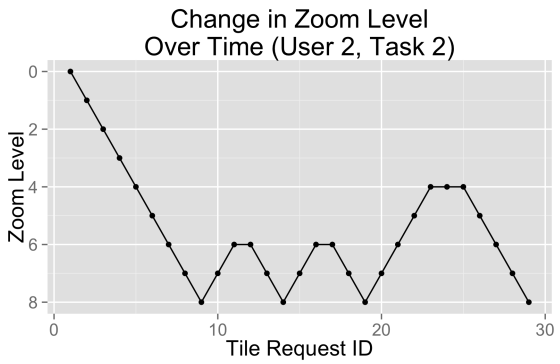


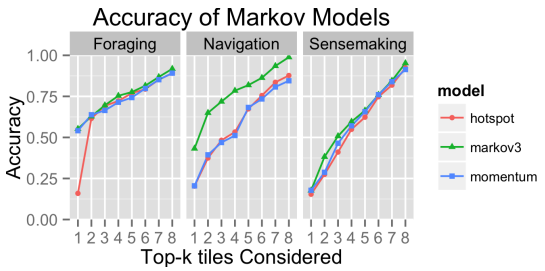
Figure 4-5: Change in zoom level per request as study participant 2 completed task 2.

downward slope corresponds to the user moving from a coarser to a more detailed zoom level; an upward slope corresponds to the reverse; and a flat line (*i.e.,* slope of 0) to the user panning to tiles at the same zoom level.

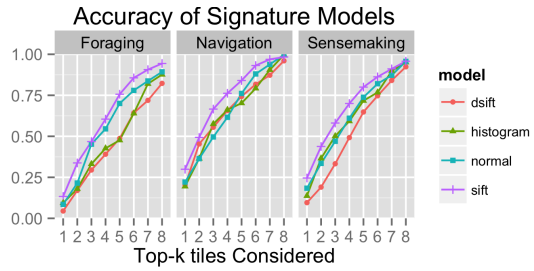
We see that the user alternates between zooming out to a coarser zoom level, and zooming into more detailed zoom levels. We know that the coarser views were used to locate snow, and the high-resolution views to find specific tiles that satisfied task 2 (hence the four tile requests specifically at zoom level 8).

We see in Figure 4-5 that this user's behavior corresponds directly to the three analysis phases described in Section 4.2.3. The user's return to coarser views near the top of Figure 4-5 correspond to the user returning to the Foraging phase (*e.g.,* request ID's 20 to 23). The user's zooms down to the bottom half of the plot correspond to the user moving to the Sensemaking phase, as they searched for individual tiles to complete the task. Furthermore, we found that 13 out of 18 users exhibited this same general exploration behavior throughout their participation in the study. 16 out of 18 users exhibited this behavior during 2 or more tasks. Furthermore, we found that only 57 out of the 1390 total requests made in the study were not described adequately by our exploration model.

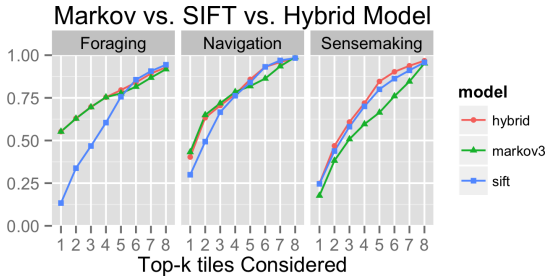
Therefore, we conclude that our three analysis phases provide an accurate classification of how the vast majority of users actually explored our NDSI MODIS dataset.



(a) **Accuracy of our best AB model (Markov3) and existing models.**



(b) **Accuracy of the four signatures in our SB model.**



(c) **Accuracy of our final engine (hybrid) and our best individual models.**

Figure 4-6: Accuracy of our AB model, SB model, and final predictor (*i.e.,* hybrid model).

4.4.4 Evaluating the 2D Predictor

Now that we have established that our three analysis phases provide a comprehensive labeling scheme for user behavior, we move on to evaluating our two-level predictor. In particular, we evaluated each level of our predictor separately, and then compared the complete predictor to the existing techniques described in our experimental setup.

At the top level of our predictor, we measured how accurately we could predict the user's current exploration phase. At the bottom level, we measured the accuracy of each recommendation model, with respect to each analysis phase.

The following experiments apply *leave-one-out cross validation* [53], a common cross-validation technique for evaluating user study data. For each user, the models were trained on the trace data of the other 17 out of 18 participants, and tested on the trace data from the remaining participant that was removed from the training set. After evaluating each user individually, we averaged the results across all users to produce our final accuracy calculations.

Predicting the User's Current Analysis Phase

The goal was to measure how accurately we could predict the user's current analysis phase. To build a training and testing set for this experiment, we manually labeled each request in our request logs with one of our 3 analysis phases. Figure 4-4b shows the distribution of phase labels. We see that users spent noticeably less time in the Foraging phase for tasks 2 and 3 (*i.e.,* looking for new ROI's), which is consistent with our user study observations.

To test our SVM classifier, we performed leave-one-out cross validation (see above), where all requests for the corresponding user were placed in the test dataset, and the remaining requests were placed in the training set. Training the classifier took less than one second. We found that our overall accuracy across all users was 82%. For some users, we could predict the current analysis phase with 90% accuracy or higher.

Accuracy of Recommendation Models

To validate the accuracy of our individual recommenders, we conducted two sets of experiments, where we: (1) compared the accuracy of our AB recommender to existing techniques, and (2) measured the prediction accuracy of our SB recommender separately for each of our four tile signatures. The goal of these experiments was two-fold. First, we wanted to find the phases where existing techniques performed well, and where there was room for improvement. Second, we wanted to test whether our AB and SB models excelled in accuracy for their intended analysis phases. To do this, we evaluated how accurately our individual models could predict the user's next move, for each analysis phase.

Action-Based (AB) Model: To evaluate the impact of history length on our AB recommender, we implemented a separate Markov chain for $n = 2$ to $n = 10$, which we refer to as Markov2 through Markov10, respectively. Each Markov chain only took milliseconds to train. We found that $n = 2$ was too small, and resulted in worse accuracy. Otherwise, we found negligible improvements in accuracy for lengths beyond $n = 3$, and thus found $n = 3$ (*i.e.,* Markov3) to be most efficient Markov chain for our AB model.

Figure 4-6a shows the prediction accuracy of our AB model compared to the Momentum and Hotspot models, with increasing values of k. Note that k represents the total space

(in tiles) that each model was given for predictions (see Section 4.4.2 for more information). In Figure 4-6a, we see that for the Foraging and Sensemaking phases, our AB model matches the performance of existing techniques for all values of k. Furthermore, we found that our AB model achieves significantly higher accuracy during the Navigation phase for all values of k. This validates our decision to use the AB model as the primary model for predicting the Navigation phase.

Signature-Based (SB) Model: Figure 4-6b shows the accuracy of each of our individual signatures, with respect to analysis phase. To do this, we created four separate recommendation models, one per signature. Amongst our signatures, we found that the SIFT signature provided the best overall accuracy. We expected a machine vision feature like SIFT to perform well, because users are comparing images when they analyze MODIS tiles.

We found that the denseSIFT signature did not perform as well as SIFT. denseSIFT performs worse because it matches entire images, whereas SIFT only matches small regions of an image. Here, relevant visualizations will contain clusters of orange snow pixels, but will not look similar otherwise. For example, the Rockies will look very different from the Andes, but they will both contain clusters of snow (orange) pixels. Thus, many relevant tiles will not appear to be similar with regards to the denseSIFT signature.

Evaluating the Final 2D Predictor

We used the accuracy results for our phase predictor and best individual recommendation models as inputs to our final two-level predictor. Our predictor only incorporated two recommenders, the AB recommender with $n = 3$ (*i.e.,* Markov3) and the SIFT SB recommender. Note that we updated our original allocation strategies based on our observed accuracy results. When the Sensemaking phase is predicted, our model always fetches predictions from our SB model only. Otherwise, our final model fetches the first 4 predictions from the AB model (or less if $k < 4$), and then starts fetching predictions from the SB model if $k > 4$.

Figure 4-6c shows that our final predictor successfully combined the strengths of our two best prediction models. It was able to match the accuracy of the best recommender

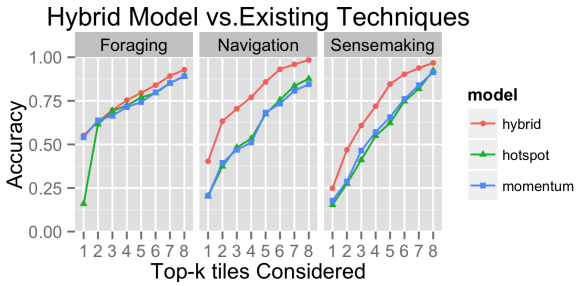


Figure 4-7: Accuracy of the hybrid model compared to existing techniques.

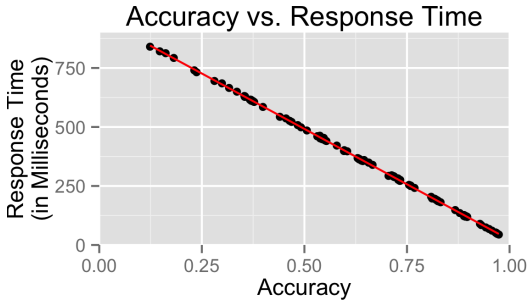


Figure 4-8: Plot of average response time given prefetch accuracy, for all models and fetch sizes (linear regression: Adj R^2=0.99985, Intercept=961.33, Slope=-939.08, P=1.1704e-242.

for each analysis phase, resulting in better overall accuracy than any individual recommendation model. We also compared our final predictor to existing techniques, shown in Figure 4-7. We see that for the Foraging phase, our predictor performs as well (if not better) than existing techniques. For the Navigation phase, we achieve up to 25% better prediction accuracy. Similarly for the Sensemaking phase, we see a consistent 10-18% improvement in accuracy.

4.4.5 Latency

We used the same setup from our accuracy experiments to measure latency. To measure the latency for each tile request, we recorded the time at which the client sent the request to the middleware, as well as the time at which the requested tile was received by the client. We calculated the resulting latency by taking the difference between these two measurements. On a cache hit, the middleware was able to retrieve the tile from main memory, allowing Sculpin to send an immediate response. On a cache miss, the middleware was forced to issue a query to SciDB to retrieve the missing tile, which was slower. On average, the middleware took 19.5 ms to send tiles for a cache hit, and 984.0 ms for a cache miss.

To evaluate our claim that accuracy dictates latency, we plotted the relationship between prefetching accuracy and average response time (*i.e.,* average latency), shown in Figure 4-8. Accuracy and response times were plotted for all models and fetch sizes. We see a strong linear relationship between accuracy and response time, where a 1% increase in accuracy corresponded to a 10ms decrease in average response time (adjusted $R^2 = 0.99985$). Given this constant accuracy-latency factor, we found that the higher prediction accuracy of our hybrid algorithm translates to a time savings of 150-250ms per tile request, when compared with existing prefetching techniques. The difference in latency is plotted in Figure 4-9, where we calculated the average response times for three models. We found that our hybrid model reduced response times by more than 50% for $k \geq 5$, compared with existing techniques.

This latency evaluation indicates that Sculpin provides significantly better performance over not only traditional systems (*i.e.,* exploration systems without prefetching), but also existing systems that enable prefetching (*e.g.,* [27, 17, 63]). Specifically, as shown in Figure 4-9, with a prefetch size of 5 tiles ($k = 5$), our system demonstrates a 430% improvement over traditional systems (*i.e.,* average latency of 185ms vs. 984ms), and 88% over existing prefetching techniques (average latency of 185 ms vs. 349 ms for Momentum, and 360 ms for Hotspot).

In addition, Sculpin provides a much more fluid and interactive user experience than traditional (no prefetching) systems. As shown in HCI literature, a 1 second interaction delay is at the limit of a user's sensory memory [15]. Delays greater than 1 second make users feel like they cannot navigate the interface freely [72, 62]. In this regard, traditional systems (*i.e.,* a constant latency of 1 second per request) are not considered interactive by HCI standards.

In contrast, Sculpin remains highly interactive during most of the user's interactions with the interface, with only 19.5ms of delay per tile request. As shown in Figure 4-7, with a fetch size of 5 tiles ($k = 5$), the prediction algorithm succeeds the vast majority of the time (82% of the time), making a cache miss (and the full 1 second delay) an infrequent event. Thus our techniques allow systems with limited main memory resources (*e.g.,* less than 10MB of prefetching space per user) to operate at interactive speeds, so that many

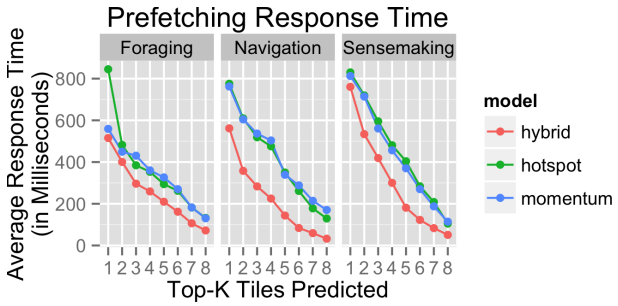


Figure 4-9: Average prefetching response times for hybrid model and existing techniques.

users can actively navigate the data freely and in parallel.

4.5 Summary

In this chapter, we presented the multidimensional prediction framework developed in Sculpin, which consists of two layers: a) an inner layer of predictors, where each predictor is responsible for making predictions for a single 2D visualization; and b) an outer vis selector layer, which efficiently allocates pre-fetching space across multiple predictors to support multiple visualizations in the Sculpin user interface. We presented results from a user study we conducted, where 18 domain experts used Sculpin to explore NASA MODIS satellite imagery data. We tested the performance of our 2D predictor design using traces recorded from our user study, and presented accuracy results showing that when exploring a single visualization in Sculpin, our predictor provides: (1) significant accuracy improvements over existing prediction techniques (up to 25% higher accuracy); and (2) dramatic latency improvements over current non-prefetching systems (430% improvement in latency), and existing prediction techniques (88% improvement in latency). We show the performance of our full prediction engine in Chapter 5.

Chapter 5

Efficient Pre-Computation and Caching of Data Tiles in Sculpin

5.1 Introduction

Through predictive pre-fetching, Sculpin is able to significantly reduce interaction latencies as the user explores her data (*i.e.,* making panning and zooming interactions fast–500ms or less), thereby achieving our first performance goal. However, pre-fetching alone fails to address the two remaining performance goals in Sculpin: reducing materialization latency, or the time spent preparing the raw data for visualization (Goal 2); and reducing the disk space consumed by data tiles (Goal 3).

In this chapter, we explain how the remaining two goals can be achieved through one simple idea: compute fewer data tiles (or execute fewer queries on the DBMS). This is because the scope of the user's query is generally much larger than what the user ultimately explores. By computing only a small fraction of data tiles for any given user query, we can simultaneously shrink both the materialization latency (Goal 2) and the storage requirements (Goal 3). To do this, we exploit the hierarchical nature of exploratory browsing: users start exploring at coarse-grained zoom levels, and only zoom in when they see something interesting. Virtually all users will explore the coarsest zoom levels. However, exploration at the finer-grained zoom levels will vary significantly by user. Therefore, if we focus on pre-computing tiles on only the coarsest zoom levels, we will cover the majority

of requested tiles for the query.

We then pair our offline computation strategy (*i.e.,* pre-compute tiles on the coarsest zoom levels first) with a predictive pre-computation strategy to maintain low interaction latency in the online case (*i.e.,* Goal 1). To do this, we identify where the user will move next, then compute these tiles just ahead of the user as she explores. Sculpin already computes the tiles from the coarsest zoom levels offline. These tiles are also the most expensive tiles to compute, because tiles at coarse zoom levels span a larger portion of the underlying dataset. As such, Sculpin has already eliminated the risk of computing expensive tiles while the user explores. Thus in the online case, the only tiles left to compute are tiles at the finest-grained zoom levels. Each of these tiles spans only a small piece of the entire dataset, making them faster to compute, and thus making predictive pre-computation more effective.

To further reduce interaction latency (addressing Goal 1), Sculpin employs two kinds of visualization-focused caching optimizations. To the best of our knowledge, Sculpin is the first system to utilize multi-level caching optimizations for general-purpose exploratory browsing. Sculpin applies two browsing-specific cache replacement policies to ensure that only the least useful tiles are removed from Sculpin's tile caches: 1) a navigation-based policy that evicts tiles that the user cannot easily navigate to; and 2) a ranking-based policy that considers how likely tiles are to be requested by the user, given her recent interactions. We also study how two different cache coordination protocols affect cache misses across Sculpin's three storage layers: coordination to ensure complete overlap between caches (mutual inclusion); and coordination to ensure no overlap between caches (mutual exclusion).

To evaluate our pre-computation and caching techniques, we conducted a second user study with 20 earth science researchers exploring satellite sensor data. We found that Sculpin provides significant improvements in both the materialization latency (380% improvement), and disk space used to store data tiles (370% improvement), while also maintaining low interaction latency (*i.e.,* average response time of 490ms or less). In summary, we make the following contributions:

1. We propose a unified design that reduces both interaction and materialization laten-

cies. The key idea is to only materialize the data tiles that will actually be viewed by the user, which represents a small fraction of the entire dataset. We then utilize multiple caches across both the client and server to more efficiently manage materialized tiles.

2. To simultaneously reduce materialization latency and storage requirements, we propose a two-stage materialization process: materialize the tiles on the coarsest zoom levels offline, before the user starts to explore. Then materialize any missing tiles just ahead of the user as she explores by utilizing the Sculpin prediction framework (see Chapter 4 for details on the prediction framework).
3. To ensure that Sculpin maintains low interaction latency, we incorporate caching across both the client and server, and propose new cache coordination protocols and cache replacement policies to maximize usage of all available cache space.
4. We evaluate our pre-computation and caching techniques in a second user study with 20 real users exploring real-world data, and we find that Sculpin provides significant improvements in materialization latency and storage requirements, while simultaneously achieving low interaction latency.

5.2 Tile Builder

A critical performance goal for Sculpin is to reduce materialization latency in the system. In Sculpin, this takes the form of reducing the time spent building data tiles offline, before the user starts exploring. However, reducing materialization latency must also be balanced with reducing interaction latency (our first performance goal). To achieve this, the Tile Builder spreads the work of building tiles between an offline process (*i.e.,* before the user explores) and an online process (*i.e.,* while the user is actively exploring a visualization). In this section we explain the steps to the offline and online tile building processes.

5.2.1 Building Tiles Offline

The offline tile-building process is designed to target both performance goal 2 (reduce materialization latency) and performance goal 3 (reduce disk storage requirements). We

achieve both goals with one simple technique: build as few tiles as possible. This technique is driven by one key insight: the scope of the user's query is significantly larger than what she actually explores. As such, the vast majority of data tiles will ultimately be ignored by the user. For example, for each task in our user study, our participants only explored 2% of the data tiles (see Section 5.4.2 for more details on our user study). If we can identify ahead of time which tiles are unlikely to be requested by the user, we can ignore these tiles completely during the offline tile-building process. Using this approach, we save not only the time required to build these tiles, but also the disk space required to store them.

To identify which tiles to build and which to ignore, we exploit the hierarchical nature of exploratory browsing. When users explore through a detail-on-demand interface, they begin their exploration at the coarsest-grained zoom levels, and only zoom in when they spot something interesting in the data. As such, the vast majority of tiles at the coarsest zoom levels are routinely explored by users. In contrast, significantly fewer tiles are explored at the finest-grained zoom levels, where users are only zooming into small, targeted regions. Therefore, if we focus on building tiles at the coarsest zoom levels, we will cover the vast majority of tiles explored by users. By targeting core behavioral patterns, our approach can be applied to any exploratory browsing system with a detail-on-demand interface.

Given that only the coarsest zoom levels will be built, the key decision that must be made during the offline build process is how many zoom levels to build. To do this, the Tile Builder considers the impact of ignoring a particular zoom level on *interaction latency* in the system. For example, if the user requests a tile at the coarsest zoom level, and this tile has not been built, she could be waiting 12 seconds or longer for this tile to appear on the screen (timing data from our experiments is provided in Section 5.4.3). Unfortunately, data tiles at the coarsest zoom levels also take the most time to build, because these tiles span significantly more of the dataset.

To combat high interaction latency, the Tile Builder considers the average time required to build a single tile at each zoom level, which we refer to as the *build estimate* for this zoom level. To do this, the Tile builder randomly selects 100 tiles from each zoom level, executes the corresponding queries, and measures the time taken to execute each query (*i.e.,* the *cost*

of each tile). The resulting tile costs are then averaged to compute the final build estimate. Sculpin then traverses each zoom level, from coarsest to finest. For each zoom level with a build estimate that is larger than a user-defined *build threshold* parameter (one second, by default) the Tile Builder builds and stores the tiles corresponding to this zoom level. Given that building any of these tiles online could have a catastrophic impact on the interaction latency, tiles that are built offline can never be evicted from the builder cache.

5.2.2 Building Tiles Online

Given that only a subset of tiles are built offline, an online build process is needed to anticipate when a user will visit tiles that are not yet built. Thus, the goal of the online build process is to predict where the user will explore next, and build the corresponding tiles, if necessary. To achieve this, the online build process uses the predictions from the Prediction Framework to identify which tiles to build ahead of time and store in the builder cache.

When the user requests a tile that has not been built, the Tile Builder issues the corresponding query to the DBMS, and returns the result to the Prediction Framework.

5.3 Caching Optimizations

To further reduce the impact of our tile building techniques on interaction latency, we developed a unified caching architecture in Sculpin that spans both the client and server. Though several systems include a single basic cache for storing pre-fetched tiles [17, 9, 27], no existing exploratory browsing systems utilize multiple caches, nor do they implement cache management strategies that exploit knowledge of how users explore array data. Smart eviction policies can reduce cache miss rates by evicting tiles from the cache that the user is unlikely to explore, making space for recommendations from the Prediction Framework. Cache coordination protocols support better overall cache utilization across the entire system. Here we propose two visualization-aware cache replacement policies, and two tile-based cache coordination protocols to further reduce interaction latency in Sculpin.

5.3.1 Cache Replacement Policies

Here, we describe the policies we employ to evict irrelevant tiles from our tile caches.

Ranking-Based Eviction: This policy labels tiles based on the likelihood that they will be requested by the user in the future. To label cached tiles, a separate call is made to the Prediction Framework, where the current state of the cache (*i.e.,* the ID's of the tiles currently stored in the cache) is passed as input. The Prediction framework returns an ordered list containing a subset of the tiles, representing the tiles that are likely to be requested by the user in the future. This ranked list is computed by running the predictors, and comparing the resulting prediction output to the current cache state. Any cached tiles that are not recommended by the predictors are selected for eviction. The remaining tiles are then prioritized based on their assigned ranking in the prediction output. Sculpin then evicts the non-recommended tiles, in LRU order. However, if the Prediction Framework believes that every tile in the cache is likely to be predicted, then this policy defaults to LRU.

Navigation-Based Eviction: This policy exploits knowledge of how users navigate in pan-zoom interfaces: if a cached tile is far away from the user's current location, it can be more safely evicted than tiles near the current location. As such, this policy requires a predefined distance threshold as input, where the distance from the user's current location to a given tile is measured in the number of user interactions (*i.e.,* pans or zooms) required for the user to see this tile in the interface. The default threshold is one interaction. When the cache is full, this policy evicts any tiles outside the given distance threshold. This measure applies to both tiles within the same zoom level, as well as tiles on adjacent zoom levels.

When all cached tiles are within the distance threshold, this policy defaults to LRU. Note that we use the LRU policy as a baseline for evaluating our other cache replacement policies. We evaluate the performance of our cache replacement policies in Section 5.4.5.

5.3.2 Cache Coordination Protocols

Previous work focuses on a single cache when managing data tiles [17, 9, 27]. However in Sculpin, the client and server each have their own caches for tile management. Here, we describe two cache coordination protocols for actively managing data tiles across multiple caches. The protocols are as follows:

Mutual Exclusion None of the caches share overlap in tiles (*i.e.,* if a tile is cached in the client, it cannot be cached in the server).

Mutual Inclusion All caches share complete overlap in tiles (*i.e.,* if a tile is cached in the client, it must also be cached in the server).

To enforce the protocols, cache updates are sent in one direction: from the client to the server, then from the Prediction Framework to the Tile Builder. Each update contains a list of tile ID's, representing the contents of the caches. Thus, the Prediction Framework receives updates that only include the client-side main memory cache, and the Tile Builder receives updates that include both the client-side and server-side main memory caches. The receiver of a cache update message is responsible for changing its cache state to match the state of the sender. In the case of Mutual Exclusion, the receiver of the update evicts any cached tiles that are on the input list. In the case of Mutual Inclusion, the receiver retrieves and caches as many tiles on the list as possible (*i.e.,* attempts to mimic the state of the sender's cache). If the receiver has a larger cache than the sender, the protocol only applies up to the cache size of the sender. Thus, any additional tiles that are not explicitly marked by the sender can stay in the receiver's cache. Sculpin uses the Mutual Inclusion protocol by default.

Currently, Sculpin does not support hybrid protocols (*i.e.,* protocols for partial overlaps). We leave these protocols for future work.

5.4 Experiments

As stated previously, Sculpin targets three specific performance goals: 1) reduce interaction latency, 2) reduce materialization latency, and 3) reduce disk space usage. In this section,

we outline how we collected realistic user data for our performance experiments (through a user study with real-world data, explained in Sections 5.4.1 and 5.4.2), explain the specific measures we focus on to evaluate Sculpin (Section 5.4.3), and analyze Sculpin's performance across all three goals (Sections 5.4.4 through 5.4.8).

As established in Chapter 4, a system is considered to have low interaction latency if the average response time is 500ms or less. Therefore, to achieve Goal 1, Sculpin must maintain this response time requirement.

For Goal 2, we aim to reduce the materialization latency as much as possible. For example, if we can reduce materialization latency by 50%, we can enable the user to explore visualizations twice as fast. Furthermore, Goals 2 and 3 are tightly coupled: the more data that gets materialized, the more time we need to prepare it and the more space we need to store it. As such, we expect that significant improvements in materialization latency will also lead to similar improvements in storage requirements. We explain exactly how we measure both materialization latency and disk space consumed in Section 5.4.3.

5.4.1 Datasets

For these experiments, Sculpin was used to explore part of a massive NASA MODIS dataset. This dataset spanned one year of MODIS satellite sensor measurements collected in or near the US in 2014 [1]. This dataset was over 12TB in size when stored within SciDB using compression (roughly 1TB per month).

To evaluate Sculpin, we visualized two queries executed on the sensor data: a snow cover query, showing the likelihood of snow at each observed point on the earth (very similar to the query analyzed in Chapter 4); and a natural color query (described in [81]), representing how the earth looks to the human eye. We refer to the snow cover query results as the NDSI dataset, for the function we used to calculate the presence of snow [85]. The natural color query results were split across two separate analyses: tracking the progress of a hurricane along the Pacific Coast (called the Hurricane dataset), and tracking the appearance of phytoplankton blooms in the Gulf of Mexico (called the Ocean Blooms dataset). The time range of each dataset is as follows: December 12 - 25 for the snow cover dataset

(aggregated per day), September 1 - 14 for the hurricane dataset (aggregated per day), and April 1 to June 14 for the ocean blooms dataset (aggregated per week). Each dataset had 3 dimensions: latitude, longitude, and time.

5.4.2 User Study

The most effective method for evaluating interactive visualization systems is to have real people use them to complete real-world tasks. As such, we designed and conducted a user study, where study participants explored satellite sensor data using Sculpin. We selected the tasks for our user study based on our previous work on similar data (discussed in Chapter 4), as well as through collaborations with domain scientists at MIT. To ensure that participants' interactions were not influenced by delays in the system, we built all data tiles ahead of time and stored them on disk for the duration of the study, similar to the approach used in Chapter 4.

Participants

Our participants were 18 graduate students and 2 postdoctoral researchers at MIT, who actively analyze or are interested in analyzing multidimensional sensor data, such as satellite sensor data. There were 15 male and 5 female participants, ages 24 to 37. Each participant completed the study once.

Study Procedure

Each participant read and signed a consent form prior to participating in the study. Participants completed the study on a laptop that we provided. We observed every participant in-person as they completed the study. At the beginning of the study, each participant was presented with brief instructions on how to use our visualization tool, and given the opportunity to explore the interface of our tool for five minutes and were encouraged to ask questions about the task or the interface. After completing the study, participants filled out a five-minute survey about the user study interface.

User Interface

In this experiment, Sculpin's draggable interface was used[1]. An example of the user interface is provided in the Sculpin overview in Chapter 3 (Figure 3-6, a user exploring 3D natural color data). For the 3D datasets, the interface consisted of two separate visualizations: a 1D time view, and a 2D map view. The time view was a draggable line chart, showing average measurements over time: the average NDSI for the snow cover dataset, and the average intensity of "red" light for the natural color views. The 2D map view was a heatmap for the snow cover data, and a Red-Blue-Green image for the natural color view, similar to the images rendered by GIS products. Participants could interact with the time view to move forward or backward in time, or interact with the map view to explore different parts of the United States.

The interface displayed 8 different tiles at once, across both coordinated visualizations. As users manipulated the interface, requests were sent to the server to retrieve new tiles. Depending on how far the user panned in the interface, 2 to 8 tiles were retrieved from the server. 8 new tiles were always requested when a zoom interaction occurred. As participants explored each dataset, we found that the vast majority of interactions were panning interactions: 92% of all interactions for the NDSI dataset, 79% of all interactions for the Hurricane dataset, and 85% of all interactions for the Ocean Blooms dataset. The median number of tiles fetched per user interaction was 4 tiles.

Tasks

Participants completed four phases of tasks, where they explored three different satellite sensor datasets (described in Section 5.4.1), one dataset per task phase. Each phase consisted of a training subtask to get users acquainted with the dataset, and a search task to complete by analyzing the dataset. All training subtasks and search tasks asked the participant to use the visualization tool to explore satellite sensor data and answer questions about this data. To answer these questions, the user had to either search for specific visual artifacts (*e.g.,* patches of snow or a phytoplankton bloom), or to analyze specific geograph-

[1]The user study in Chapter 4 used an early prototype interface, with button-based panning interactions.

ical regions in or near the United States (*e.g.,* the Gulf of Mexico, Utah, Washington). We selected our tasks to be as realistic as possible, while ensuring that a diverse group of participants could complete the study in a reasonable amount of time.

Across all participants, we found that only a small fraction of data tiles were explored per task: 0.77% for NDSI, 0.85% for Hurricane, and 2.03% for Ocean Blooms.

5.4.3 Experimental Setup

In this section, we explain the physical setup and experimental design for our performance experiments.

Physical Setup

All experiments are performed using two servers running Ubuntu 12.04 and SciDB. One of the two servers also runs the Sculpin middleware layer. The servers each have 48GB of RAM. One server has 10TB of disk space; the other server has 20TB of space.

User Study Data

We use traces collected from our user study to conduct our performance experiments. Note that **none of the pre-computed tiles from the user study are included in these experiments**. As such, these performance experiments must also take into account the time required to compute any requested data tiles. A single trace from the study corresponds to one user completing a single task, which we refer to as a user session. We perform leave-one-out cross validation [53], which is a well-known cross validation technique used in machine learning, and an evaluation strategy employed in Chapter 4. In leave-one-out cross validation, the system is trained on traces from 19 of our 20 study participants, and tested on the traces from the participant that was left out. This process is repeated for all 20 participants, providing 20 separate results. We then average across all participants to produce our final experimental results.

A user session is an ordered list of user interactions. To evaluate a single user session, we iterate over this list of interactions in order. For each interaction, we send both the

Table 5.1: Four separate cases (0.25, 0.5, 1.0, and 1.5 seconds) for the average time to build a single tile at the cheapest zoom level (bottom row, in bold). Each case defines average tile cost for every zoom level in the datasets. The zoom levels with tile costs lower than 1 second are highlighted in yellow for each case. The last column shows the total number of tiles on each zoom level. These numbers are for the NDSI and Hurricane tasks.

NDSI, Hurricane	Tile Cost (in seconds)				
Zoom Level	Case 1	Case 2	Case 3	Case 4	Total Tiles
0	12.990	25.980	51.959	77.939	14
1	3.756	7.511	15.023	22.534	56
2	1.393	2.786	5.572	8.358	224
3	0.745	1.490	2.979	4.469	686
4	0.483	0.966	1.931	2.897	2744
5	0.355	0.709	1.418	2.127	10206
6	**0.25**	**0.5**	**1.0**	**1.5**	40824

Table 5.2: Four separate cases (0.25, 0.5, 1.0, and 1.5 seconds) for the average time to build a single tile at the cheapest zoom level (bottom row, in bold). Each case defines average tile cost for every zoom level in the datasets. The zoom levels with tile costs lower than 1 second are highlighted in yellow for each case. The last column shows the total number of tiles on each zoom level. These numbers are for the Ocean Blooms task.

Ocean Blooms	Tile Cost (in seconds)				
Zoom Level	Case 1	Case 2	Case 3	Case 4	Total Tiles
0	7.967	15.934	31.870	47.805	13
1	2.347	4.693	9.387	14.081	52
2	0.872	1.746	4.490	5.238	208
3	0.575	1.149	2.298	3.447	637
4	0.279	0.559	1.117	1.676	2548
5	**0.25**	**0.5**	**1.0**	**1.5**	9477

previous $n = 3$ interactions (*i.e.,* the interaction history) and the user's current interaction to the Sculpin client-side Tile Manager, which sends this information to the server-side middleware component. Sculpin then: 1) retrieves the tiles corresponding to the user's current interaction, and sends them to the client; then 2) sends the user's interaction history to the Prediction Framework and Tile Builder to make predictions. All of our evaluation metrics are updated per user interaction.

Evaluation Metrics

To evaluate performance goal 1 (interaction latency), we measure average response time, the metric applied in Chapter 4, as well as previous work [63, 62]. We measure response time by subtracting the time the client issued a request from the time the client received a response from the server. To calculate the average response time for a single user session, we sum the response times for this session, and divide the sum by the total number of requests.

To evaluate performance goal 2 (materialization latency), we first measure the time taken to complete the offline tile building process in Sculpin (*i.e.,* the time to prepare the user's query for visualization). We then divide this value by the total time required to build every data tile in the dataset (*i.e.,* the approach used by many exploratory browsing systems [63, 60, 9]).

To evaluate performance goal 3 (storage space consumption), we take a similar approach. We count the total number of unique tiles that were built by Sculpin within a single user session (both offline and online). We then divide this value by the total number of tiles in the dataset (*i.e.,* the number of tiles built by existing exploratory browsing systems).

Simulating Tile Building and Retrieval

In Sculpin, tiles are built by executing the corresponding array query in SciDB, and storing the result as a separate array. To fetch a tile from SciDB, Sculpin issues a fetch query (*i.e.,* a scan query for the computed tile array). Here, we describe how we calculate and measure both tile build times and tile fetch times in our experiments.

Using a single-threaded, single-disk setup, we found that the cheapest tile cost was roughly 1.5 seconds, which we refer to as the *baseline condition*. The tile costs per zoom level from this analysis are provided in the fifth column of Tables 5.1 and 5.2, labeled Case 4. The total number of tiles per zoom level is provided in column 6, labeled Total Tiles. We simulate the same analysis (shown as Cases 1-3) by scaling the compute time of the baseline condition. The purpose of these simulations is to estimate the compute time of different server configurations, such as when using multiple disks for data replication, or

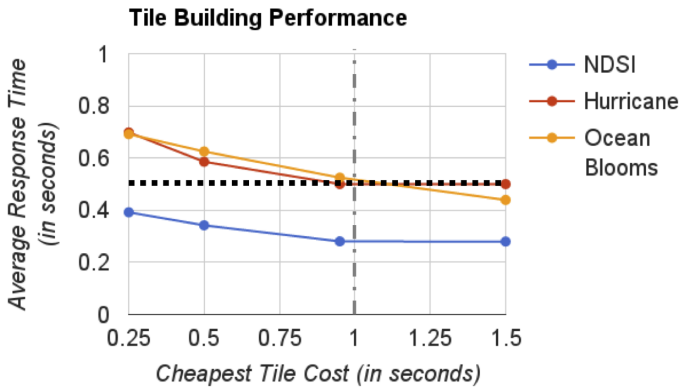


Figure 5-1: Response time results divided by task, and averaged across all users. The cost of building tiles is varied across four different cases listed in Section 5.4.3. The dotted line is our target response time (500ms).

using multi-threaded execution.

For example, to create Case 3, where the tile cost for the cheapest zoom level is 1 second, we multiply the original tile cost (1.5 seconds) by a scaling factor of $\frac{2}{3}$. Cheap zoom levels, or levels with tiles that take less than one second to build, are highlighted in yellow. Only expensive tiles are built during the offline process. Thus, each case represents a different number of tiles being built. These tile costs are applied for all experiments involving the Tile Builder (Sections 5.4.8 and 5.4.4).

To compare with previous work, we chose to also simulate fetch times. In our previous user study in Chapter 4, we found that Sculpin had an average fetch time of 984ms when retrieving pre-computed tiles from SciDB. Thus, we set our simulated fetch time to be the same.

When considering the total tiles per zoom level in Tables 5.1 and 5.2, we see that the finer grained zoom levels contain significantly more tiles. As a result, these zoom levels require more space to store, as well as more time to compute.

5.4.4 Tile Builder Evaluation

The Tile Builder is designed to target our last two performance goals: reduce materialization latency, and reduce disk storage requirements. We measure materialization latency as the time spent building tiles offline. We measure disk storage as the number of tiles built during a single user session (both offline and online). Our focus in this set of ex-

Table 5.3: The fraction of time spent building tiles offline, recorded for each task and each of the four tile building cases described in Section 5.4.3.

	Fraction of Time Spent Building Tiles Offline (in %).		
Tile Building Cases	NDSI	Hurricane	Ocean Blooms
Case 4 (1.5)	100	100	100
Case 3 (1.0)	23.8	23.8	26.3
Case 2 (0.5)	2.5	2.5	6.8
Case 1 (0.25)	0.7	0.7	1.4

periments is to compare Sculpin's tile building strategy (build only tiles from the coarsest zoom levels offline) with the strategy of existing exploratory browsing systems (build everything offline). To do this, we calculate relative measures for the time spent building tiles (Sculpin time divided by total time), and number of tiles built (Sculpin count divided by total tiles). Each metric is measured separately for each task and each tile building case in Section 5.4.3.

Experimental Parameters: The offline build threshold is set to 1 second. Therefore, any zoom levels with average tile cost less than 1.0 will be ignored by the offline tile building process. These zoom levels are highlighted in yellow in Tables 5.1 and 5.2. In the online build process, we re-use the Prediction Framework to decide which tiles to pre-compute after each user request, similar to the pre-fetching process described in Section 5.4.7. In this case, we only allow the Prediction Framework to recommend 16 tiles to pre-compute after each user request, which we found empirically to be the most effective parameter setting for the online build process. Note that only a small fraction of tiles are ever pre-computed online, since the vast majority of tiles requested by participants are at the coarsest zoom levels, which are already computed offline. In these experiments, our caching optimizations are *turned off*, so we can measure the exact contribution of our tile building techniques to the overall performance of Sculpin.

Materialization Latency Results: Table 5.3 shows the fraction of time spent building tiles offline. To calculate this, we count the number of tiles Sculpin built per zoom level, multiply each count by its corresponding tile cost cost from Tables 5.1 and 5.2, and sum the results. Then we divide the resulting sums by the total processing time required to build

every tile in the dataset (*i.e.,* the approach of previous work [63, 60, 9]). In 3 out of 4 cases, the bottom zoom level is "cheap," and can be ignored during the offline process (*i.e.,* none of the bottom zoom level is materialized offline). By ignoring this one zoom level, Sculpin already provides a 3.8x improvement in materialization latency. Furthermore in Case 1, nearly all zoom levels are "cheap" (*i.e.,* most tiles in the dataset can be built in under one second). Here, Sculpin can exploit this fact and build just two or three zoom levels within the dataset, leading to a 71x improvement in materialization latency for Case 1. In Case 4, every zoom level must be built offline, representing the worst case scenario for Sculpin. As we can see, Sculpin matches the performance of existing work in this case.

Disk Space Results: Table 5.4 shows the fraction of tiles built offline by the Tile Builder, where this fraction is calculated for each task by dividing the number of tiles that Sculpin built by the total number of tiles in the corresponding dataset. We see that the Tile Builder provides similar improvements in storage space: 62x improvement for Case 1, 13x for Case 2, 3.7x for Case 3, and comparable performance in Case 4.

Impact on Interaction Latency: Here, we measure the average response time in Sculpin (*i.e.,* the interaction latency), when using only our tile building optimizations (*i.e.,* caching is still turned off). In Figure 5-1, we see that when the cheapest zoom level has an average tile cost of 1.5 seconds (Case 4), we get the best average response time from Sculpin: 280ms for NDSI, 500ms for Hurricane, and 439ms for Ocean Blooms. This is expected, because the Tile Builder builds every tile offline here. However, because every tile is built in advance, this case results in the worst results for materialization latency and disk usage (*i.e.,* no improvement over existing systems). As we decrease the number of zoom levels built offline, we see dramatic reductions in storage space and materialization latency. However, this also increases the average response time, since the Tile Builder will now have to build more tiles online. The worse case for response times is Case 1, where the cheapest zoom level has an average tile cost of 250 milliseconds. In this case, the Tile Builder only builds 2 of the zoom levels in advance, and Sculpin has average response times of 391ms for NDSI, 698ms for Hurricane, and 691 milliseconds for Ocean Blooms. The response times for NDSI are significantly better (*i.e.,* lower) due to having high prediction accuracy from the Prediction Framework for this task (pre-fetching is always on by default). Thus,

Table 5.4: The fraction of tiles built offline by the Tile Builder, recorded for each task and each of the four tile building cases described in Section 5.4.3.

	Fraction of Tiles Built (in %)		
Tile Building Cases	NDSI	Hurricane	Ocean Blooms
Case 4 (1.5)	100	100	100
Case 3 (1.0)	25.4	25.4	27.0
Case 2 (0.5)	1.8	1.8	7.6
Case 1 (0.25)	0.6	0.6	1.6

if we use only our tile building optimizations, we see a tradeoff: when we spend less time building tiles offline, we increase the average response time. In the next section, we explain how we combine our tile building optimizations with our caching optimizations to achieve all 3 of our performance goals.

5.4.5 Caching Evaluation

While our tile building optimizations provide dramatic improvements in materialization latency and disk storage requirements, they alone are insufficient to address all three of our performance goals. We developed our caching optimizations specifically to target our first performance goal: to maintain low interaction latency (*i.e.,* average response times of 500ms or less). In this section, we compare our cache replacement policies to LRU, test our cache coordination protocols, and report on Sculpin's average response times when combining our tile building and caching optimizations.

Cache Replacement Policies

Experimental Parameters: In this experiment, only the server-side main memory cache is enabled, and tile building optimizations are turned off (*i.e.,* every tile is built offline). We vary the number of tiles Sculpin is allowed to cache from the user's recent requests, ranging from 4 to 128 tiles. We evaluated average response times with all three of our cache replacement policies: LRU, Navigation-based, and Rank-based.

Interaction Latency Results: We found that Navigation-based and Rank-based were both notably better than LRU across all three tasks. The response time results per cache

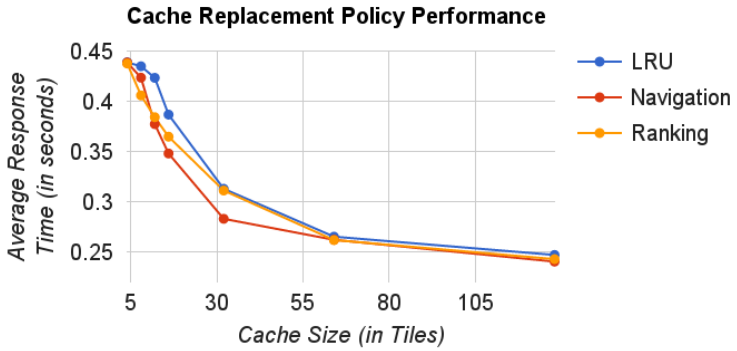


Figure 5-2: Average response time results for the Ocean Blooms task, when caching 4-128 tiles in Sculpin, calculated separately for each cache replacement strategy. Similar results were observed for all three tasks.

replacement policy are provided in Figure 5-2. We observed very similar results for all three tasks, however to conserve space we only show results for the Ocean Blooms task. We found that the Navigation-based cache replacement policy can provide a 46ms reduction in response time compared to LRU, and the Rank-based policy can provide a 32 ms reduction. However, as the cache increases in size, Sculpin is able to store more of the user's recent requests, making sophisticated cache eviction strategies less critical. Given that the Navigation-based policy is generally better than the Rank-based policy, we only use the Navigation-based policy for our later experiments.

Figure 5-3 shows a steady decrease in response times as the size of the server-side main memory cache increases from 4 to 32 tiles. However, cache sizes larger than 32 tiles seemed to provide only a marginal benefit. For example, increasing the cache size from 32 to 128 tiles only reduced response times by 17ms for the NDSI task. Therefore for the remainder of our experiments, we set the size of our tile caches to 32 tiles.

Cache Coordination Protocols

Experimental Parameters: In this experiment, we enable caching in both the server-side main memory cache (managed by the Prediction Framework) and the builder cache (managed by the Tile Builder). Our tile building optimizations are now turned on (*i.e.,* only a fraction of data tiles are built offline). In the previous experiment, we found that a cache size of 32 tiles to be most effective. Therefore, each cache has space to store 32 tiles from

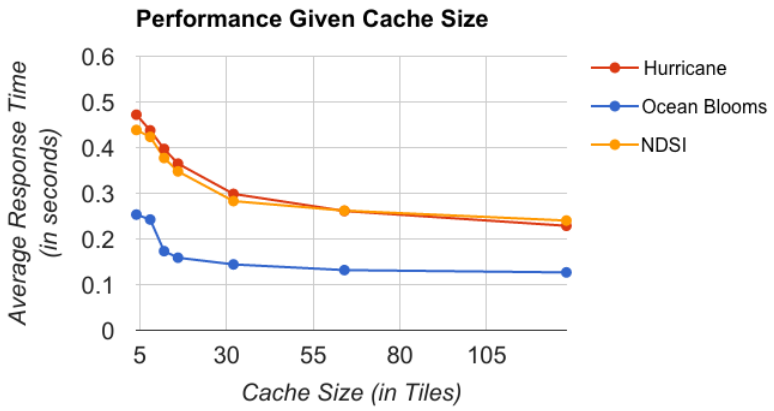


Figure 5-3: Average response time results for all 3 tasks, when caching 4-128 tiles in Sculpin, using the Navigation cache replacement policy.

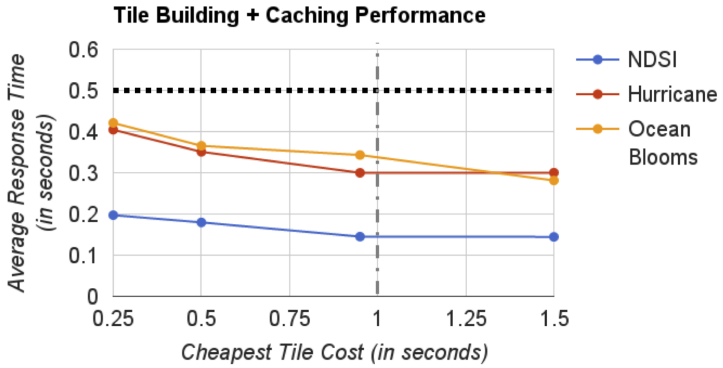


Figure 5-4: Average response time results for each tile building case (0.25, 0.5, 1.0 and 1.5 seconds), and each dataset, when both the tile building and caching optimizations are enabled in Sculpin. The vertical dashed line represents Sculpin's default build threshold of 1 second. The horizontal dashed line represents the interactivity threshold of 500ms.

the user's recent requests. The Navigation-based cache replacement policy is used for both caches due to its superior performance. We experimented with our two cache coordination protocols: Mutual Inclusion and Mutual Exclusion.

Interaction Latency Results: We found little performance difference between the two and therefore only report the results using the Mutual Inclusion coordination protocol. Figure 5-4 shows the new average response time results. For all four tile building cases, our caching techniques result in a consistent 200-300ms reduction in response time across all three datasets. Even when compared directly with existing techniques, where every tile is pre-computed offline (*i.e.,* Case 4), our caching optimizations still provide at least: a 56% improvement in performance for the Ocean Blooms dataset, a 66% improvement for the Hurricane dataset, and a 93% improvement for the NDSI dataset. As a result, we see that for all four tile building cases, Sculpin is able to provide interactive response times (*i.e.,*

Table 5.5: Average response time (in seconds), with pre-fetching, tile building and caching enabled. The client-side cache was either turned on (with size 32 tiles), or turned off. Two separate clients were tested: a laptop connected to a public wireless network, and the server running Sculpin.

Machine	Client Caching	NDSI	Hurricane	Ocean Blooms
Laptop	On	0.173	0.444	0.483
	Off	0.203	0.519	0.598
Server	On	0.136	0.282	0.305
	Off	0.138	0.301	0.326

500ms or less).

5.4.6 Impact of Client-Side Caching

In this experiment, we ran our end-to-end system with tile building case 3 (the 1 second case), and Mutual Inclusion as the default coordination protocol. We evaluated average response times when client-side caching was both on and off, and using two different client machines: a laptop using the public wireless network at MIT, and the server running the Sculpin back-end. The results are provided in Table 5.5. We found that when the client and server are co-located, client-side caching provides only a marginal improvement in performance: 2ms improvement for NDSI, 19ms for Hurricane, and 21ms for Ocean Blooms. However, when using a laptop on a public network, we observe a noticeable improvement in performance when client-side caching is turned on: 30ms improvement for NDSI, 75ms for Hurricane, and 115ms for Ocean Blooms. Furthermore, we found that using client-side caching brought the average response time within interactive bounds for this case (*i.e.,* under 500ms).

5.4.7 Prediction Framework Evaluation

Predictive data pre-fetching is the most common technique in exploratory browsing for reducing interaction latency [9, 17, 27, 26]. As such, Sculpin incorporates pre-fetching as a key optimization technique that is always in use. However, in addition to competitive performance, Sculpin also provides a new approach to efficiently pre-fetching data tiles in the multidimensional case. Here, we report briefly on our results.

Comparing With the 2D Case: We compared Sculpin's full multidimensional framework with the performance of a single 2D predictor, which represents the best prediction technique developed thus far for exploratory browsing in the 2D case (*e.g.,* panning and zooming in a latitude-longitude map) [9]. Specifically, we compare Sculpin's prediction accuracy and average response times for the NDSI dataset to the 2D snow cover case evaluated in our previous user study (see Chapter 4 for more details). Note that no existing systems currently support prediction beyond the 2D case (*e.g.,* panning and zooming through latitude, longitude, *and* time).

Experimental Parameters: To ensure a fair comparison is made with our previous study, we pre-compute all data tiles in advance for this experiment. To study the impact of total pre-fetching space on performance, we measure prediction accuracy as we increase pre-fetching space (in tiles). For each user study task, we perform leave-one-out cross validation for five different pre-fetch sizes: 4, 8, 16, 22, and 32 tiles pre-fetched before each user interaction. We observed in the user study that the median number of tiles requested per user interaction was 4 tiles. Thus, pre-fetching 4 tiles is equivalent to predicting 1 of the user's possible interactions, 8 tiles to 2 interactions, and so on. The cross validation results for each task and pre-fetch size are aggregated across all users.

Prediction Results: With a pre-fetch size of 16 tiles (*i.e.,* predicting 4 of the user's 10 possible directional moves), Sculpin achieves 72% prediction accuracy (283ms response time). This is very close to the prediction results observed for the 2D case, which achieved 74% accuracy when pre-fetching 4 of the user's 9 possible directional moves (250ms response time). We found that increasing the pre-fetching size beyond 16 tiles provided only marginal improvements in performance.

5.4.8 End-to-End System Evaluation

Here, we summarize Sculpin's end-to-end performance on the 1 second tile building case (Case 3). To do this, we measure all three of our performance goals in a realistic user environment.

Experimental Parameters: For this experiment, our tile building optimizations and

caching optimizations (with all three caches enabled) are turned on. A laptop connected to a public wireless network was used as the client.

Interaction Latency Results: Using a laptop connected to a public wireless network as the client machine, Sculpin provided an average response time of 490ms or less for all three datasets, achieving performance goal 1 (maintaining low interaction latency).

Materialization Latency Results: With existing exploratory browsing systems [63, 60, 9], users must wait for a slow offline build process to complete before seeing any visualizations of their queries. This is problematic when the user wants to explore a new query quickly. Thus, to enable the user to see query results faster (and thus iterate over her analyses faster), Sculpin reduces the *materialization latency*, or the time spent preparing the user's query for visualization. To measure this, we calculate the time Sculpin spends building data tiles offline, and divide this time by the total time required to compute every data tile in advance (*i.e.,* the approach of exiting systems). We found that Sculpin provided a 420% improvement in materialization latency for the NDSI and and Hurricane datasets. For the Ocean Blooms dataset, Sculpin provided a 380% improvement (down to 150 minutes from 572 minutes). Thus, users can visualize queries in Sculpin four times as fast as other systems, achieving performance goal 2 (reducing materialization latency).

Storage Results: The client-side cache was 32MB in size, and the server-side main memory cache was 48MB in size, representing a small main-memory footprint for client and server-side caching. The builder cache contained roughly 14GB of data for the NDSI and Hurricane tasks, which provided a 393% improvement in space usage (originally 54.7GB). Similarly, the builder cache contained 3.5GB of data for the Ocean Blooms task (originally 12.9GB), resulting in a 374% improvement in space usage. Thus, Sculpin achieves performance goal 3 (reducing storage requirements).

5.5 Summary

In this chapter, we presented two new techniques in Sculpin, incremental tile building and visualization-aware caching, which are used in conjunction with data-prefetching to reduce both materialization latency (*i.e.,* the time required to prepare the underlying data for

visualization) and disk space usage (*i.e.,* total tiles built by Sculpin). To evaluate our precomputation and caching techniques, we conducted a second user study with 20 domain scientists exploring satellite sensor data. We found that Sculpin maintains low interaction latency (average response time of 500ms or less), while also significantly reducing materialization latency (380% improvement) and disk space consumed by materialized results (370% improvement). Thus, Sculpin reduces the time and space required to develop interactive visualizations, enabling users to quickly and iteratively analyze complex analyses over large multidimensional datasets.

Chapter 6

Related Work

The techniques developed in Sculpin span several areas of research: 1) latency analysis for visual exploration systems; 2) data resolution reduction for efficient visualization of massive datasets; 3) analysis of user interaction logs to extract and model user behavior; and 4) interface design techniques and systems optimizations to support efficient multidimensional data exploration. Here, we highlight the projects and techniques in these areas that are most relevant to Sculpin.

6.1 Latency Analysis for Visual Exploration Systems

A critical component of this thesis is to better understand how latency alters users' exploration behaviors in search-based data analytics tasks. Here, we highlight work in psychology, HCI and visualization that inspired our experimental design and analysis.

6.1.1 Psychology of Searching

Early exploration of human information processing uncovered two complimentary processes at work in the context of visual search: automatic detection (*i.e.,* committing the objects to be found to long-term memory prior to performing the search task) and controlled search (*i.e.,* using a methodical, exhaustive and often serial process to identify the objects to be found) [88, 90]. Automatic detection was demonstrated to become sharper

with repeated exposure, developing from a consistent mapping between stimuli and response over multiple examples. In contrast, *controlled search* is a more methodical serial comparison behavior which requires active attention by the subject. It has been hypothesized that controlled processing operations such as this make extensive use of short-term memory [88], and are frequently disrupted by automatic detection [90].

6.1.2 Latency in VIS and HCI

Disengagement due to latency in visualization is a frequently noted phenomenon, and addressing latency has led to advancements in approximate and pre-cached query results, which we discuss in Section 6.2. The negative effect of high latency on exploratory analysis was explicitly measured in [62] and [108], with both works concluding that users perform better with lower latency. Research in Human Computer Interaction has found that user preference is highly skewed towards sub-100ms latency in direct touch input [71, 46], yet latency has little effect on the higher level tasks involved in a real-time strategy game [22, 89]. The stratification of higher-order and lower-order tasks agrees with the evidence found in psychology research that suggests different speeds (and thus different latency "thresholds") for different user actions.

6.1.3 Influencing a User's Analysis Behavior

By making subtle changes to the task setting and comparing the behavior of a test and control group, it can be shown that behavior is a function of the change in setting. In one such experiment, participants were shown different pictures before being asked to do a creativity task, and it was found that participants shown a picture of a happy baby performed better than those shown a hammer [57]. Priming the user's emotions prior to viewing a visualization has been shown to effect perception [39], while priming locus of control effected user strategy in a visual search task [75]. Behavioral manipulations can also occur continuously throughout the entire task. The user's attention in a digital view can also be manipulated with subtle flickers [100], modulations [99], or changing regional saliency [8]. In Chapter 2, we manipulate the latency of each interaction to infer the functional relationship

between latency and visual search strategy for different analysis tasks.

6.2 Data Resolution Reduction

In an effort to support fast rendering of massive datasets, many systems apply aggregation or sampling techniques to reduce the amount of data to be rendered on the screen. This technique is also known as *resolution reduction* [10]. Here we discuss the recent advancements in each of these areas to support scalable visualization.

6.2.1 Aggregation

Aggregation techniques are generally applied across the entire dataset, and thus are often considered an expensive operation that must be completed prior to rendering visualizations. As such, aggregation is generally applied offline, before the user starts to explore her data. However, an offline aggregation process can be beneficial because it condenses potentially massive query results into a much smaller data range, resulting in significantly fewer datapoints to be rendered in the final visualization. One example of applying aggregation to DBMS query results is to calculate a windowed average across the results (*e.g.,* as explained in Section 3.2), such that the width of the aggregation windows match the range of available pixels on the user's laptop, a technique utilized in our prior work [10]. Here, we discuss two general aggregation techniques applied in visualization systems: hierarchical aggregation, and specialized data cubes. Sculpin leverages the aggregation techniques proposed in previous work to provide concise visualizations of query results.

Hierarchical Aggregation

Elmqvist and Fekete [32] developed an abstraction for applying hierarchical aggregation to efficiently process and visualize spatial (and spatiotemporal) data. The focus of their work was to visually simplify aggregate visualizations of massive datasets by reducing clutter, allowing aggregate visualizations such as treemaps to scale up to larger dataset sizes.

Data Cubes

Data cubes [35] are a specialized data structure for supporting efficient execution of aggregate analytical operations over large datasets. They have been studied and optimized for many years within the database community, the most relevant techniques to Sculpin being efficient data cube materialization [5, 21, 69]. However, raw data cubes are ill-suited to efficient data processing for visualization applications. In response, two data-cube variants have emerged to make data cubes more effective for fast data processing for visualization: data tiles [63], which are also utilized in Sculpin, and nanocubes [60] (and the more recent hashedcubes [76]). Nanocubes are a specialized form of data cubes for spatiotemporal datasets that directly incorporate hierarchical aggregation results. Using nanocubes, Lins *et al.* are able to support extremely fast visualization of large spatiotemporal datasets, assuming that the entire nanocubes structure can fit in memory on the user's client machine.

However, a major drawback to the nanocubes data structure (and data cube-like structures in general) is the amount of space required to store these data structures. Sculpin significantly reduces the amount of space consumed by data tiles by incrementally building the data tiles, and carefully selecting only the most critical data tiles for pre-computation.

6.2.2 Sampling

Efficient sampling of large datasets has been an active area of database research for many years, and continues to gain interest in both the database and visualization communities. Here, we highlight the major sampling strategies that have been developed, and their applications to visualization. Two major strategies are considered when building samples for visualization: offline sampling, where the sample is built at data load time, and online or progressive sampling, where the sample is built on the fly (*i.e.,* at query execution time).

We note here that sampling is an orthogonal optimization technique that can be applied alongside the optimizations proposed in Sculpin. As such, a complete survey of sampling techniques is outside the scope of this thesis[1].

[1] We defer to recent work in this area to provide a more comprehensive review [4, 25, 58].

Offline Sampling

A variety of query sampling engines have been developed to efficiently build samples at data load time, which can be used later to quickly render new visualizations. These techniques focus on stratified sampling [64], and either rely on prior knowledge of the queries to be executed over the sample (i.e., the query workload) [4, 19, 2, 3, 91, 73], or build samples without knowledge of the query workload [2, 3, 7].

These techniques are used infrequently for visualization use cases, as they are less flexible for ad-hoc analysis and visualization. Users often visualize and then discard new query results, making long-term samples of these results to be of limited use.

Online Sampling

Hellerstein et al. (Control [41]) and Fisher et al. (Sample Action [33]) use online aggregation to visualize individual aggregates (*e.g.,* a sum or a count) from increasingly large samples over time. Kamat et al. (DICE [47]) and Crotty et al. (A-WARE [23]) use progressive sampling to enable sub-second response times for a larger set of visualization types.

Online aggregation techniques can also be applied in a distributed fashion to improve performance. One such example is the VisReduce system developed by Im *et al.* [44], which utilizes MapReduce-style data processing techniques to create visualizations.

Li et al. propose a new technique to efficiently sample database joins using random walks [58], providing a new avenue of research for progressive visualization of complex queries over massive datasets.

Sampling Parameters

However, in all sampling cases, one must know when a sample is "large enough" to provide an accurate visualization of query results. In the case of progressive sampling, this problem falls on the shoulders of the user, who must then decide for themselves when they have seen enough information. However, it would be significantly more helpful to users if visualization systems could determine appropriate sample sizes automatically. To this end, recent work has studied how to set these parameters for specific visualization use cases.

For example, Kim et al. developed an algorithm to identify the smallest sample size required to render a bar chart such that the bars are correctly ordered by height [51]. Park et al. calculate minimum size samples for scatterplot and map charts [77] using a loss function that ensures that the visual differences between the "true" visualization and sampled visualization are within a given threshold.

6.3 Analysis of User Interactions

Sculpin's optimization strategy relies on the ability to record, collect and mine user interaction logs to develop a better understanding of user behavior. Several projects have also explored techniques to extract behavioral information and train behavioral models from raw interaction logs. Here, we discuss a number of techniques for visualizing, analyzing and mining insights from interaction logs.

6.3.1 Interaction Log Analysis

Wei et al. developed a visualization system to facilitate visual exploration and analysis of web clickstream data [102]. Heer et al. explore various design decisions for developing interfaces for interaction log (or history) analysis, and propose new techniques for efficiently rendering and navigating visualization histories [40]. Guo *et al.* developed a new framework for mining interaction logs from text analysis interfaces [37]. Using this framework, they can quantitatively measure correlations between specific interaction features and insight generation and characteristics. Brown et al. studied user search patterns while users performed a search task involving the classic Where's Waldo image puzzles [14]. By training machine learning techniques on the interaction logs from the study, they could accurately predict whether a participant would finish the Waldo puzzle quickly or slowly, and even infer specific personality factors for this participant such as extroversion and locus of control.

6.3.2 Pre-fetching and Prediction

Several systems use data pre-fetching to reduce user wait times (*i.e.,* system latency) when exploring database queries, using a single visualization to explore 2D datasets. Doshi et al. (Xmdv Tool [27, 26]) and Chan et al. (ATLAS [17]) propose simple direction-based prediction techniques for pre-fetching data. Lee et al. [56] and Cetintemel et al. [16] propose using more sophisticated behavioral models, such as Markov chains and SVM models.

In Sculpin, we include more data-driven prediction algorithms, and propose a general-purpose approach for combining existing prediction techniques. We also extend previous work to support data pre-fetching for multiple visualizations and multidimensional datasets.

Furthermore, despite the clear importance of efficient data management in supporting interactive visualization [107], existing visualization systems that utilize pre-fetching still ignore the impact of multi-level data caching on reducing response times. In a related area, Li et al. study multi-level caching specifically for maps [59]. We extend existing work by proposing cache replacement and coordination strategies that exploit knowledge of user exploration behavior in general-purpose exploratory browsing systems.

6.3.3 Sensemaking Models

The analysis model developed in Chapter 4 is based on the sensemaking model presented by Pirolli and Card [80]. In this work, Pirolli and Card describe the following flow for data analysis: a *foraging loop*, where one is focused on collecting and filtering information, followed by a *sensemaking loop*, where one constructs a theory or hypothesis based on the information collected. We adopt these ideas in our analysis model as separate Foraging and Sensemaking phases (see Chapter 4 for more details). However, a subset of interaction patterns that we observed in our user studies are not well represented by the Pirolli and Card sensemaking model. For example, the navigation-focused interactions we observed, such as zooming in to a particular ROI, or zooming out to the coarsest zoom level after performing an in-depth analysis of an ROI, do not seem to match the foraging or sensemaking loops of this model. As such, we incorporated a separate Navigation analysis phase into

our analysis model to support these interaction patterns.

6.4 Multidimensional Visualization Exploration

Supporting visual exploration of multidimensional data is becoming increasingly important. As such, several visualization systems have been developed to allow users to explore multidimensional datasets. van den Elzen and van Wijk [97], Key *et al.* [50], Vartak *et al.* [98] and Wongsuphasawat *et al.* [106] developed interfaces that provide thumbnails of recommended visualizations (or small-multiples) that users can click on to explore a new visualization of their data, along various data dimensions. However, these systems are designed for small-scale data analysis. Kamat *et al.* [47] developed an interface to support exploration along different data cube dimensions via a drop-down query menu. However, forcing the user to interact with a query menu prevents the user from directly manipulating the current visualization to explore her data. Thus, these systems lack the ability to support fluid in-depth exploration of large-scale datasets. Nanocubes [60], hashedcubes [76] and imMens [63] support multidimensional datasets, but still have dataset size limitations. In contrast, Sculpin supports in-depth exploration of massive arrays through direct manipulation of multiple visualizations within a coordinated view visualization design.

Chapter 7

Future Work

In this chapter, we propose several future directions for scalable visual analytics. We first discuss direct opportunities to further improve the performance of visual exploration tools like Sculpin. We then turn to a broader discussion of how visual exploration tools can potentially shape the future of big data analytics and data science.

7.1 Making Visual Exploration More Effective

Sculpin provides significant improvements over existing techniques, but there are some drawbacks to the current design. Here, we discuss the limitations to Sculpin, and propose several avenues for future work: support for more flexible exploration, support for more dataset types, finer-grained incremental tile building, and support for more diverse perceptual measures of exploratory browsing systems.

7.1.1 Supporting more interaction types

Sculpin only lets users pan and zoom, and prevents users from performing other kinds of exploration, such as searching for tiles containing specific characteristics (*e.g.,* tiles with no cloud cover), or jumping to specific locations. An interesting area for future work is to extend the tile-based data model (and existing prediction techniques) to support a wider variety of interactions.

7.1.2 Supporting more dataset types

Sculpin is designed primarily for exploring large, multidimensional matrices. However, scientists explore many types of large datasets, such as social network data and medical data, which have completely different structures. Important future work for systems like Sculpin must include new data structures and optimization techniques to support a wider variety of dataset types.

For example, our proposed tiling scheme works well for arrays. However, when considering other types of data (e.g., social graphs or patient health records), it is unclear how to map these datasets to tiles. We plan to develop a general-purpose tiling mechanism for relational datasets. We also plan to study how tiling parameters affect performance for array and non-array datasets.

Similarly, we manually identified four signatures for our SB recommender, and found that SIFT works best for the NDSI dataset (see Chapter 4 for more details). However, other features may be more appropriate for different datasets. For example, counting outliers or computing linear correlations may work well for prefetching time series data. We plan to build a general-purpose signature toolbox with more of these signatures, and plan to extend Sculpin to learn what signatures work best for a given dataset automatically. To evaluate Sculpin's new tiling scheme and signature toolbox, we plan to conduct a user study on non-array-based datasets. For example, we plan to support the MIMIC II medical dataset, which provides hospital data recorded for thousands of patients and many data types (e.g., unstructured text, tabular, and array data).

7.1.3 Finer-grained tile building

Sculpin only builds a subset of all possible data tiles. However, to ensure that the user never has to wait for an expensive tile to be computed while she explores, Sculpin takes an overly conservative approach, and builds entire zoom levels at a time (see Chapter 5 for more details). It is unlikely that every tile needs to be computed for most zoom levels, including expensive ones. To accurately identify which expensive tiles can be safely avoided, we need better metrics for measuring the risk of building (or not building) specific tiles. We

also need new techniques for predicting which tiles are likely to be requested by the user *before* she starts exploring (*i.e.,* without any information about her exploration behavior).

In addition, our prediction framework does not currently take into account potential optimizations within a multi-user scheme. For example, it is unclear how to partition the middleware cache to make predictions for multiple users exploring different datasets, or how to share data between users exploring the same dataset. We plan to extend our architecture to manage coordinated predictions and caching across multiple users.

7.1.4 More Perceptual Measures

A hybrid evaluation strategy was adopted throughout this thesis, where we collected real behavioral data through user studies, and then retroactively evaluated Sculpin's performance using traces collected from these studies. This strategy allowed us to study how humans explore massive sensor datasets, and also to gauge how effective our optimizations are in supporting fast data exploration.

However, as shown in our results from our first study in Chapter 2, our current evaluation method may have blind spots, and we need new evaluation methods to concretely measure the impact of poor system performance on exploratory browsing in a standardized way. Specifically, new perceptual measures are needed to better evaluate both the positive and negative effects of various optimization techniques. We suggested a technique for measuring the impact of latency in an exploratory browsing system by gauging user preferences for low versus high-latency data regions. Similar measures for identifying bias in the conclusions drawn by the user, and the number of "useful" insights reached by the user are potential measures to be developed for the future.

Ultimately, we aim to better understand the dynamics of the partnership formed between data scientists and exploratory browsing systems. The system reacts to the user's behavior, but the user also reacts to the system's behavior. How can we better characterize this bidirectional effect, and study how the dynamics change when new system optimizations and features are introduced to the system?

7.2 Making Visual Exploration More Relevant

Interactive data visualization and exploration at scale is an exciting area of research with many new possible research directions to pursue. Here, we provide a sketch of some promising directions for the fields of data science and big data analytics, with respect to data visualization.

7.2.1 Generalizing Data Exploration at Scale

Many user interfaces have been designed to support data exploration in a variety of areas, including biology, medicine, urban planning, earth science, physics, social science, and astronomy. However, no one has explored what this means for the fields of data management and visualization. Specifically, it is unclear how many unique interactions exist across all these interfaces. Interesting future work would be to evaluate whether and how these disparate interaction types could be unified across domains. A critical piece of this future direction will be to develop new optimizations that translate across disparate data structures and semantics.

7.2.2 Automated Recommendations for Data Analysis

Data analysis is a complex process, where users have to make many decisions about how to best analyze their data. This thesis focuses on making individual analysis iterations fast using exploratory browsing systems. As such, this thesis focuses on improving the system's performance. However, the human's performance also plays an important role in efficiency. Specifically, if the user makes smart choices about how to analyze her data, she can reach her goals with less effort (i.e., fewer and faster iterations). Therefore, it is critical to consider a more holistic view of what it means for the data analysis process to be efficient, and to develop optimizations accordingly.

One such opportunity is to extend behavioral modeling and interaction prediction to provide automated recommendations. These recommendations would provide real-time feedback to the user as she analyzes her dataset, so she can immediately redirect the course

of her analysis to better match her goals. In the case of Sculpin, the system could recommend new data regions of potential interest to the user, given the regions she recently visited in the past. These recommendations can potentially reduce the number of interactions that the user performs within a single analysis iteration, but more importantly, it might help the user identify new patterns in the data that she had previously missed. Taking this a step further, the system can also help direct the course of the user's analysis using speculative execution. Specifically, if the system can predict which analyses the user will want to perform in the future (e.g., what DBMS query the user will write next), then it can speculatively execute these analyses during the user's current analysis iteration, and recommend them to the user for her next iteration.

7.2.3 Collaborative Data Analysis Systems

Human collaboration is a natural and common facet to data analysis and exploration. For example, it is common for several users to work together to formulate an initial hypothesis or research direction for analyzing a new dataset. However, most existing data exploration systems only support single-user experiences. To provide the most effective support in all aspects of the data analysis process, exploratory browsing systems must support multiple users exploring datasets together.

The first step to supporting collaborative exploration is to support multi-user exploration, where many users are exploring different datasets at the same time. Here, the challenges involve identifying when resources should be shared between users, and how to best distribute resources across all users. Given that user exploration behaviors change over time, exploratory browsing systems also need to be able to adapt their resource allocations on the fly. Thus an important component to this trajectory of future work will be to develop new techniques to support a diverse set of exploration patterns evolving simultaneously.

Chapter 8

Conclusions

The focus of this thesis has been to provide users (*e.g.,* data scientists) with a fast, fluid interaction experience when using exploratory browsing tools to visualize and explore massive array data. As such, we focused on reducing system latency, or the time required by the system to respond to a user's actions. To do this, we first characterized how users choose and switch between high level search strategies when faced with system latency, and then developed three separate optimization techniques to reduce system latency in exploratory browsing systems, implemented in our system Sculpin.

Through our initial user study in Chapter 2, we demonstrated how system latency can have clear negative effects on the user's ability to quickly and easily analyze her data. Our participants explored image collages, where each image in the collage represented an image "tile" within the interface. We simulated system latencies by inserting a delay before making image tiles appear on the screen. From our study, we found that when there are long delays in when and where tiles appear on the screen, the system can lead the user to draw certain conclusions about the dataset. In the case of our study, each user was given multiple target pictures find in the collage. The results of our study demonstrate that users showed clear preferences for certain target images within the collage when system latencies were introduced to the interface. Given these results, it is clear that latency in the system can cause users to bias their actions, hindering their ability to freely and objectively analyze their data. As such, it is critical to minimize the impact of system latency on users, which we propose to do by reducing the amount of latency observed by users in the system.

We then define the two forms of system latency that impact users: *materialization latency*, or the time required to prepare and materialize query results in the DBMS for visualization; and *interaction latency*, or the time required to fetch data from the DBMS in response to a user pan or zoom interaction.

In response, we developed Sculpin, a general-purpose system designed for interactive and iterative exploration of large, multidimensional datasets. Sculpin employs a client-server architecture, where the user interacts with a lightweight client-side interface, and the data to be explored is retrieved from a back-end server running a DBMS. For efficient data processing and retrieval, Sculpin utilizes a tile-based data model, where query results are partitioned into fixed-width blocks, or data tiles. We developed three optimization techniques in Sculpin to address both materialization and interaction latency. These techniques were implemented within a modular optimization layer that interacts with the client-side user interface and server-side DBMS. Sculpin employs three separate techniques in the optimization layer to improve performance: 1) a multidimensional prediction framework, to anticipate the user's future interactions and pre-fetch the corresponding data tiles ahead of time to reduce interaction latency; 2) incremental tile pre-computaiton techniques, to reduce the number of tiles that need to be computed before the user's visualization can be rendered in the client-side interface; and 3) visualization-aware caching techniques to further boost performance as the user explores her dataset.

We conducted two separate user studies to evaluate Sculpin, where in total 38 earth scientists used Sculpin to explore NASA MODIS satellite sensor data. We tested the performance of Sculpin using traces recorded from both of our user studies. We presented performance results showing that our prediction framework provides: (1) significant accuracy improvements over existing prediction techniques (up to 25% higher accuracy); and (2) dramatic latency improvements over current non-prefetching systems (430% improvement in latency), and existing prediction techniques (88% improvement in latency). Using all three performance optimizations together, we found that Sculpin maintains low interaction latency (average response time of 490ms or less), while also improving the materialization latency (380% improvement) and space consumed by materialized query results (370% improvement).

Thus, Sculpin reduces the overall system latency and storage requirements for developing interactive visualizations, enabling users to quickly and iteratively analyze complex analyses over massive, multidimensional datasets.

Appendix A

Additional MODIS Data Processing

Here, we outline the necessary steps to duplicate the analysis queries we used in Chapter 5, using data from the NASA MODIS.

To calculate the snow cover dataset, we follow the same approach as in Chapter 4: join the two arrays corresponding to the bands for visible light (band 4) and short-wave infrared light (band 6) measurements, and calculate the normalized difference snow index (or NDSI) using the reflectance measurement from these bands. The NDSI ranges from -1 (no snow) to +1 (high likelihood of snow), and is calculated for each SciDB array cell. The NDSI is as follows, where S_4 is the band 4 array, and S_6 is the band 6 array in SciDB:

$$(S_4.ref \text{ - } S_6.ref) \text{ / } (S_4.ref + S_6.ref).$$

Note that it is straightforward to translate this function into a user defined function in SciDB [81] (see Chapter 4 for the exact SciDB query used to calculate the NDSI).

To calculate the natural color dataset, we follow the approach of Planthaber *et al.* [81], where we join the bands representing red, blue, and green visible light (bands 1, 4, and 3, respectively):

$$\texttt{join(join(}S_1\texttt{,}S_4\texttt{),}S_3\texttt{).}$$

The result can then be sent to the front-end to be rendered as a natural color view.

Filtering for Cloud Cover

Removing cloud cover is a common filtering step, applied before calculating snow cover. We use a simplified version of the approach proposed by Song *et al.* to filter for clouds [92],

which involves three different MODIS bands: a water vapor absorption band (band 26), and two cloud/land boundary detection bands (bands 1 and 2). Radiance measurements are used for band 26, and reflectance measurements for bands 1 and 2. To determine whether a particular MODIS measurement (*i.e.,* array cell in the MODIS arrays) contains cloud cover, at least one of the following conditions must be true: $b_{26} \leq 6.0$; $b_1 \leq 0.25$; $b_2/b_1 > 1.6$; or $b_2/b_1 < 0.8$.

It is straightforward to combine these four conditions into a single function that returns a boolean as output. Thus, this cloud-filtering function can also be implemented as a user defined function with three inputs (*i.e.,* the cloud filtering bands).

To apply the cloud filtering function to the above queries, we perform array joins as necessary to combine the cloud filtering bands, then apply the cloud filtering udf as a filter:

$$\texttt{filter(join(join(}S_{26}\texttt{,}S_1\texttt{),}S_2\texttt{),}$$
$$\texttt{cloud\_udf(}S_{26}.rad\texttt{,}S_1.ref\texttt{,}S_2.ref\texttt{))}.$$

We can then join the filtered result with additional MODIS arrays (*e.g.,* the NDSI bands) to ensure that only cloud-free measurements are included in our calculations.

Bibliography

[1] LAADS Web – Search for Data Products. `https://ladsweb.nascom.nasa.gov/search/?si=Terra%20MODIS&si=Aqua%20MODIS`, 2016.

[2] Swarup Acharya, Phillip B. Gibbons, and Viswanath Poosala. Congressional samples for approximate answering of group-by queries. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 487–498, New York, NY, USA, 2000. ACM.

[3] Swarup Acharya, Phillip B. Gibbons, and Viswanath Poosala. Congressional samples for approximate answering of group-by queries. *SIGMOD Rec.*, 29(2):487–498, May 2000.

[4] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 29–42, New York, NY, USA, 2013. ACM.

[5] Fuat Akal, Klemens Böhm, and Hans-Jörg Schek. Olap query evaluation in a database cluster: A performance study on intra-query parallelism. In *East European Conference on Advances in Databases and Information Systems*, pages 218–231. Springer, 2002.

[6] Ioannis Arapakis, Xiao Bai, and B Barla Cambazoglu. Impact of response latency on user behavior in web search. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 103–112. ACM, 2014.

[7] Brian Babcock, Surajit Chaudhuri, and Gautam Das. Dynamic sample selection for approximate query processing. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 539–550, New York, NY, USA, 2003. ACM.

[8] Reynold Bailey, Ann McNamara, Nisha Sudarsanam, and Cindy Grimm. Subtle gaze direction. *ACM Transactions on Graphics (TOG)*, 28(4):100, 2009.

[9] Leilani Battle, Remco Chang, and Michael Stonebraker. Dynamic Prefetching of Data Tiles for Interactive Visualization. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 1363–1375, New York, NY, USA, 2016. ACM.

[10] Leilani Battle, Michael Stonebraker, and Remco Chang. Dynamic reduction of query result sets for interactive visualization. In *2013 IEEE International Conference on Big Data*, pages 1–8, October 2013.

[11] David J Bjornstad and Michael McKee. Making enduring choices: Uncertainty and public policy. *Energy economics*, 28(5):667–676, 2006.

[12] Narasimha Bolloju and Felix SK Leung. Assisting novice analysts in developing quality conceptual models with uml. *Communications of the ACM*, 49(7):108–112, 2006.

[13] Danah Boyd and Kate Crawford. Critical questions for big data: Provocations for a cultural, technological, and scholarly phenomenon. *Information, communication & society*, 15(5):662–679, 2012.

[14] Eli T Brown, Alvitta Ottley, Helen Zhao, Quan Lin, Richard Souvenir, Alex Endert, and Remco Chang. Finding waldo: Learning about users from their interactions. *IEEE Transactions on visualization and computer graphics*, 20(12):1663–1672, 2014.

[15] Stuart K. Card, George G. Robertson, and Jock D. Mackinlay. The Information Visualizer, an Information Workspace. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, pages 181–186, New York, NY, USA, 1991. ACM.

[16] Ugur Cetintemel, Mitch Cherniack, Justin DeBrabant, Yanlei Diao, Kyriaki Dimitriadou, Alexander Kalinin, and Olga Papaemmanouil. Query Steering for Interactive Data Exploration. In *6th Biennial Conference on Innovative Data Systems Research*, Asilomar, CA, USA, January 2013.

[17] Sye-Min Chan, Ling Xiao, J. Gerth, and P. Hanrahan. Maintaining interactivity while exploring massive time series. In *IEEE Symposium on Visual Analytics Science and Technology, 2008. VAST '08*, pages 59–66, October 2008.

[18] Remco Chang, Caroline Ziemkiewicz, Tera Marie Green, and William Ribarsky. Defining insight for visual analytics. *IEEE Computer Graphics and Applications*, 29(2):14–17, 2009.

[19] Surajit Chaudhuri, Gautam Das, and Vivek Narasayya. Optimized stratified sampling for approximate query processing. *ACM Trans. Database Syst.*, 32(2), June 2007.

[20] Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–394, October 1999.

[21] Ying Chen, Andrew Rau-Chaplin, F Dehne, Todd Eavis, D Green, and Elankayer Sithirasenan. cgmolap: Efficient parallel generation and querying of terabyte size

rolap data cubes. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 164–164. IEEE, 2006.

[22] Mark Claypool. The effect of latency on user performance in real-time strategy games. *Computer Networks*, 49(1):52–70, 2005.

[23] Andrew Crotty, Alex Galakatos, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. The Case for Interactive Data Exploration Accelerators (IDEAs). HILDA '16, pages 11:1–11:6, New York, NY, USA, 2016. ACM.

[24] Mihaly Csikszentmihalyi. Flow and the psychology of discovery and invention. *New Yprk: Harper Collins*, 1996.

[25] Bolin Ding, Silu Huang, Surajit Chaudhuri, Kaushik Chakrabarti, and Chi Wang. Sample + Seek: Approximating Aggregates with Distribution Precision Guarantee. SIGMOD '16, pages 679–694, New York, NY, USA, 2016. ACM.

[26] Punit R. Doshi, Geraldine E. Rosario, Elke A. Rundensteiner, and Matthew O. Ward. A Strategy Selection Framework for Adaptive Prefetching in Data Visualization. In *Proceedings of the 15th International Conference on Scientific and Statistical Database Management*, SSDBM '03, pages 107–116, Washington, DC, USA, 2003. IEEE Computer Society.

[27] Punit R. Doshi, Elke A. Rundensteiner, and Matthew O. Ward. Prefetching for Visual Data Exploration. In *Proceedings of the Eighth International Conference on Database Systems for Advanced Applications*, DASFAA '03, pages 195–, Washington, DC, USA, 2003. IEEE Computer Society.

[28] Pierre Dragicevic. Fair statistical communication in hci. In *Modern Statistical Methods for HCI*, pages 291–330. Springer International Publishing, 2016.

[29] Trafton Drew, Melissa L-H Võ, and Jeremy M Wolfe. The invisible gorilla strikes again sustained inattentional blindness in expert observers. *Psychological science*, page 0956797613479386, 2013.

[30] David H Ebenbach and Colleen F Moore. Incomplete information, inferences, and individual differences: The case of environmental judgments. *Organizational behavior and human decision processes*, 81(1):1–27, 2000.

[31] Anne Edland and Ola Svenson. Judgment and decision making under time pressure. In *Time pressure and stress in human judgment and decision making*, pages 27–40. Springer, 1993.

[32] N. Elmqvist and J. Fekete. Hierarchical Aggregation for Information Visualization: Overview, Techniques, and Design Guidelines. *IEEE Transactions on Visualization and Computer Graphics*, 16(3):439–454, May 2010.

[33] Danyel Fisher, Igor Popov, Steven Drucker, and m.c. schraefel. Trust Me, I'M Partially Right: Incremental Visualization Lets Analysts Explore Large Datasets Faster. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 1673–1682, New York, NY, USA, 2012. ACM.

[34] Kenneth G Furton and Lawrence J Myers. The scientific foundation and efficacy of the use of canines as chemical detectors for explosives. *Talanta*, 54(3):487–500, 2001.

[35] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data mining and knowledge discovery*, 1(1):29–53, 1997.

[36] Tera Marie Green, William Ribarsky, and Brian Fisher. Building and applying a human cognition model for visual analytics. *Information visualization*, 8(1):1–13, 2009.

[37] Hua Guo, Steven R Gomez, Caroline Ziemkiewicz, and David H Laidlaw. A Case Study Using Visualization Interaction Logs and Insight Metrics to Understand How Analysts Arrive at Insights. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):51–60, January 2016.

[38] Anzu Hakone, Lane Harrison, Alvitta Ottley, Nathan Winters, Caitlin Gutheil, Paul KJ Han, and Remco Chang. Proact: Iterative design of a patient-centered visualization for effective prostate cancer health risk communication. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):601–610, 2017.

[39] Lane Harrison, Drew Skau, Steven Franconeri, Aidong Lu, and Remco Chang. Influencing visual judgment through affective priming. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2949–2958. ACM, 2013.

[40] Jeffrey Heer, Jock Mackinlay, Chris Stolte, and Maneesh Agrawala. Graphical histories for visualization: Supporting analysis, communication, and evaluation. *IEEE transactions on visualization and computer graphics*, 14(6), 2008.

[41] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. Online Aggregation. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, SIGMOD '97, pages 171–182, New York, NY, USA, 1997. ACM.

[42] Richards J Heuer. *Psychology of intelligence analysis*. Lulu.com, 1999.

[43] Christoph Hölscher and Gerhard Strube. Web search behavior of internet experts and newbies. *Computer networks*, 33(1):337–346, 2000.

[44] Jean-Francois Im, Felix Giguere Villegas, and Michael J. McGuffin. VisReduce: Fast and responsive incremental information visualization of large datasets. pages 25–32. IEEE, October 2013.

[45] Jessica L James. *Mobile dating in the digital age: computer-mediated communication and relationship building on Tinder*. PhD thesis, Texas State University, 2015.

[46] Ricardo Jota, Albert Ng, Paul Dietz, and Daniel Wigdor. How fast is fast enough?: a study of the effects of latency in direct-touch pointing tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2291–2300. ACM, 2013.

[47] Niranjan Kamat, Prasanth Jayachandran, Karthik Tunga, and Arnab Nandi. Distributed and interactive cube exploration. In *2014 IEEE 30th International Conference on Data Engineering (ICDE)*, pages 472–483, March 2014.

[48] Sean Kandel, Andreas Paepcke, Joseph M Hellerstein, and Jeffrey Heer. Enterprise data analysis and visualization: An interview study. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2917–2926, 2012.

[49] Diane Kelly and Colleen Cool. The effects of topic familiarity on information search behavior. In *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, pages 74–75. ACM, 2002.

[50] Alicia Key, Bill Howe, Daniel Perry, and Cecilia Aragon. VizDeck: Self-organizing Dashboards for Visual Analytics. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 681–684, New York, NY, USA, 2012. ACM.

[51] Albert Kim, Eric Blais, Aditya Parameswaran, Piotr Indyk, Sam Madden, and Ronitt Rubinfeld. Rapid sampling for visualizations with ordering guarantees. *Proc. VLDB Endow.*, 8(5):521–532, January 2015.

[52] Ran Kivetz and Itamar Simonson. The effects of incomplete information on consumer choice. *Journal of Marketing Research*, 37(4):427–448, 2000.

[53] Ron Kohavi. A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[54] Christof Körner, Heiner Gertzen, Clemens Bettinger, and Dietrich Albert. Comparative judgments with missing information: A regression and process tracing analysis. *Acta psychologica*, 125(1):66–84, 2007.

[55] Alan Krupnick, Richard Morgenstern, Michael Batz, Peter Nelson, Dallas Burtraw, Jhih-Shyang Shih, and Michael McWilliams. *Not a sure thing: Making regulatory choices under uncertainty*. Resources for the Future Washington, DC, 2006.

[56] Dong Ho Lee, Jung Sup Kim, Soo Duk Kim, Ki-Chang Kim, Yoo-Sung Kim, and Jaehyun Park. Adaptation of a Neighbor Selection Markov Chain for Prefetching Tiled Web GIS Data. In *Proceedings of the Second International Conference on*

Advances in Information Systems, ADVIS '02, pages 213–222, London, UK, UK, 2002. Springer-Verlag.

[57] Sheena Lewis, Mira Dontcheva, and Elizabeth Gerber. Affective computational priming and creativity. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 735–744. ACM, 2011.

[58] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. Wander join: Online aggregation via random walks. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 615–629, New York, NY, USA, 2016. ACM.

[59] Xiaoming Li, Weiping Xu, Qing Zhu, Jinxing Hu, Han Hu, and Yeting Zhang. A Multi-Level Cache Approach for Realtime Visualization of Massive 3d GIS Data. *International Journal of 3-D Information Modeling (IJ3DIM)*, 1(3):37, July 2012.

[60] Lauro Lins, James T Klosowski, and Carlos Scheidegger. Nanocubes for Real-Time Exploration of Spatiotemporal Datasets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2456–2465, December 2013.

[61] Raanan Lipshitz and Orna Strauss. Coping with uncertainty: A naturalistic decision-making analysis. *Organizational behavior and human decision processes*, 69(2):149–163, 1997.

[62] Zhicheng Liu and Jeffrey Heer. The effects of interactive latency on exploratory visual analysis. *IEEE transactions on visualization and computer graphics*, 20(12):2122–2131, 2014.

[63] Zhicheng Liu, Biye Jiang, and Jeffrey Heer. imMens: Real-time Visual Querying of Big Data. In *Proceedings of the 15th Eurographics Conference on Visualization*, EuroVis '13, pages 421–430, Aire-la-Ville, Switzerland, Switzerland, 2013. Eurographics Association.

[64] Sharon Lohr. *Sampling: Design and Analysis*. Nelson Education, 2009.

[65] I Scott MacKenzie and Colin Ware. Lag as a determinant of human performance in interactive systems. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*, pages 488–493. ACM, 1993.

[66] Alex Marsh and Kenneth Gibb. Uncertainty, expectations and behavioural aspects of housing market choices. *Housing, Theory and Society*, 28(3):215–235, 2011.

[67] Michael Meehan, Sharif Razzaque, Mary C Whitton, and Frederick P Brooks. Effect of latency on presence in stressful virtual environments. In *virtual reality, 2003. Proceedings. IEEE*, pages 141–148. IEEE, 2003.

[68] Timothy JF Mitchell, Sherry Y Chen, and Robert D Macredie. Hypermedia learning and prior knowledge: domain expertise vs. system expertise. *Journal of Computer Assisted Learning*, 21(1):53–64, 2005.

[69] Arnab Nandi, Cong Yu, Philip Bohannon, and Raghu Ramakrishnan. Distributed cube materialization on holistic measures. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 183–194. IEEE, 2011.

[70] Margaret A Neale and Gregory B Northcraft. Experience, expertise, and decision bias in negotiation: The role of strategic conceptualization. *Research on negotiation in organizations*, 2:55–75, 1990.

[71] Albert Ng, Julian Lepinski, Daniel Wigdor, Steven Sanders, and Paul Dietz. Designing for low-latency direct-touch input. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pages 453–464. ACM, 2012.

[72] Jakob Nielsen. Powers of 10: Time Scales in User Experience, October 2009.

[73] Christopher Olston, Edward Bortnikov, Khaled Elmeleegy, Flavio Junqueira, and Benjamin Reed. Interactive analysis of web-scale data. In *CIDR*, 2009.

[74] Lisa Ordonez and Lehman Benson. Decisions under time pressure: How time constraint affects risky decision making. *Organizational Behavior and Human Decision Processes*, 71(2):121–140, 1997.

[75] Alvitta Ottley, R Jordan Crouser, Caroline Ziemkiewicz, and Remco Chang. Manipulating and controlling for personality effects on visualization tasks. *Information Visualization*, 14(3):223–233, 2015.

[76] Cícero AL Pahins, Sean A Stephens, Carlos Scheidegger, and Joao LD Comba. Hashedcubes: Simple, Low Memory, Real-Time Visual Exploration of Big Data. *IEEE Transactions on Visualization and Computer Graphics*, PP(99):1–1, 2016.

[77] Yongjoo Park, Michael Cafarella, and Barzan Mozafari. Visualization-aware sampling for very large databases. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 755–766, May 2016.

[78] Adam Pauls and Dan Klein. Faster and smaller n-gram language models. HLT, pages 258–267, Stroudsburg, PA, USA, 2011.

[79] William A Pike, John Stasko, Remco Chang, and Theresa A O'connell. The science of interaction. *Information Visualization*, 8(4):263–274, 2009.

[80] Peter Pirolli and Stuart Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proc. International Conference on Intelligence Analysis*, volume 2005, pages 2–4, 2005.

[81] Gary Planthaber, Michael Stonebraker, and James Frew. EarthDB: Scalable Analysis of MODIS Data Using SciDB. In *Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*, BigSpatial '12, pages 11–19, New York, NY, USA, 2012. ACM.

[82] BAJ Reddi and RHS Carpenter. The influence of urgency on decision time. *Nature neuroscience*, 3(8):827–830, 2000.

[83] Eyal M Reingold, Neil Charness, Marc Pomplun, and Dave M Stampe. Visual span in expert chess players: Evidence from eye movements. *Psychological Science*, 12(1):48–55, 2001.

[84] William Ribarsky, Brian Fisher, and William M Pottenger. Science of analytical reasoning. *Information Visualization*, 8(4):254–262, 2009.

[85] Karl Rittger, Thomas H. Painter, and Jeff Dozier. Assessment of methods for mapping snow cover from MODIS. *Advances in Water Resources*, 51:367 – 380, 2013. 35th Year Anniversary Issue.

[86] Jonathan C. Roberts. State of the Art: Coordinated & Multiple Views in Exploratory Visualization. In *Proceedings of the Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization*, CMV '07, pages 61–71, Washington, DC, USA, 2007. IEEE Computer Society.

[87] Karen D Schenk, Nicholas P Vitalari, and K Shannon Davis. Differences between novice and expert systems analysts: What do we know and what do we do? *Journal of management information systems*, 15(1):9–50, 1998.

[88] Walter Schneider and Richard M Shiffrin. Controlled and automatic human information processing: I. detection, search, and attention. *Psychological review*, 84(1):1, 1977.

[89] Nathan Sheldon, Eric Girard, Seth Borg, Mark Claypool, and Emmanuel Agu. The effect of latency on user performance in warcraft iii. In *Proceedings of the 2nd workshop on Network and system support for games*, pages 3–14. ACM, 2003.

[90] Richard M Shiffrin and Walter Schneider. Controlled and automatic human information processing: Ii. perceptual learning, automatic attending and a general theory. *Psychological review*, 84(2):127, 1977.

[91] Lefteris Sidirourgos, Martin L Kersten, Peter A Boncz, et al. Sciborq: Scientific data management with bounds on runtime and quality. In *CIDR*, volume 11, pages 296–301, 2011.

[92] Xiaoning Song, Zhenhua Liu, and Yingshi Zhao. Cloud detection and analysis of MODIS image. In *Geoscience and Remote Sensing Symposium, 2004. IGARSS '04. Proceedings. 2004 IEEE International*, volume 4, pages 2764–2767 vol.4, September 2004.

[93] Emad Soroush, Magdalena Balazinska, Simon Krughoff, and Andrew Connolly. Efficient Iterative Processing in the SciDB Parallel Array Engine. In *Proceedings of the 27th International Conference on Scientific and Statistical Database Management*, SSDBM '15, pages 39:1–39:6, New York, NY, USA, 2015. ACM.

[94] Michael Stonebraker, Paul Brown, Alex Poliakov, and Suchi Raman. The Architecture of SciDB. In *Proceedings of the 23rd International Conference on Scientific and Statistical Database Management*, SSDBM'11, pages 1–16, Berlin, Heidelberg, 2011. Springer-Verlag.

[95] Alistair G. Sutcliffe and Neil A. M. Maiden. Analysing the novice analyst: cognitive models in software engineering. *International Journal of Man-Machine Studies*, 36(5):719–740, 1992.

[96] Rebecca Taft, Manasi Vartak, Nadathur Rajagopalan Satish, Narayanan Sundaram, Samuel Madden, and Michael Stonebraker. GenBase: A Complex Analytics Genomics Benchmark. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 177–188, New York, NY, USA, 2014. ACM.

[97] Stef van den Elzen and Jarke J. van Wijk. Small Multiples, Large Singles: A New Approach for Visual Data Exploration. In *Proceedings of the 15th Eurographics Conference on Visualization*, EuroVis '13, pages 191–200, Chichester, UK, 2013. The Eurographs Association & John Wiley & Sons, Ltd.

[98] Manasi Vartak, Sajjadur Rahman, Samuel Madden, Aditya Parameswaran, and Neoklis Polyzotis. SeeDB: Efficient Data-driven Visualization Recommendations to Support Visual Analytics. *Proc. VLDB Endow.*, 8(13):2182–2193, September 2015.

[99] Eduardo E Veas, Erick Mendez, Steven K Feiner, and Dieter Schmalstieg. Directing attention and influencing memory with visual saliency modulation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1471–1480. ACM, 2011.

[100] Manuela Waldner, Mathieu Le Muzic, Matthias Bernhard, Werner Purgathofer, and Ivan Viola. Attractive flicker–guiding attention in dynamic narrative visualizations. *IEEE transactions on visualization and computer graphics*, 20(12):2456–2465, 2014.

[101] Colin Ware and Ravin Balakrishnan. Reaching for objects in vr displays: lag and frame rate. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 1(4):331–356, 1994.

[102] Jishang Wei, Zeqian Shen, Neel Sundaresan, and Kwan-Liu Ma. Visual cluster exploration of web clickstream data. In *Visual Analytics Science and Technology (VAST), 2012 IEEE Conference on*, pages 3–12. IEEE, 2012.

[103] Steffen Werner and Bjorn Thies. Is" change blindness" attenuated by domain-specific expertise? an expert-novices comparison of change detection in football images. *Visual Cognition*, 7(1-3):163–173, 2000.

[104] Ryen W White, Susan T Dumais, and Jaime Teevan. Characterizing the influence of domain expertise on web search behavior. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 132–141. ACM, 2009.

[105] Ryen W White and Dan Morris. Investigating the querying and browsing behavior of advanced search engine users. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 255–262. ACM, 2007.

[106] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations. *IEEE Transactions on Visualization and Computer Graphics*, PP(99):1–1, 2015.

[107] Eugene Wu, Leilani Battle, and Samuel R. Madden. The Case for Data Visualization Management Systems: Vision Paper. *Proc. VLDB Endow.*, 7(10):903–906, June 2014.

[108] Emanuel Zgraggen, Alex Galakatos, Andrew Crotty, Jean-Daniel Fekete, and Tim Kraska. How progressive visualizations affect exploratory analysis. *IEEE Transactions on Visualization and Computer Graphics*, 2016.