Combining User Interaction, Speculative Query Execution and Sampling in the *DICE* System

Prasanth Jayachandran Karthik Tunga Niranjan Kamat Arnab Nandi
Computer Science & Engineering
The Ohio State University
{jayachan,tunga,kamatn,arnab}@cse.osu.edu

ABSTRACT

The interactive exploration of data cubes has become a popular application, especially over large datasets. In this paper, we present *DICE*, a combination of a novel frontend query interface and distributed aggregation backend that enables interactive cube exploration. *DICE* provides a convenient, practical alternative to the typical *offline* cube materialization strategy by allowing the user to explore *facets* of the data cube, trading off accuracy for interactive response-times, by sampling the data. We consider the time spent by the user perusing the results of their current query as an opportunity to execute and cache the most likely followup queries. The frontend presents a novel intuitive interface that allows for sampling-aware aggregations, and encourages interaction via our proposed faceted model. The design of our backend is tailored towards the low-latency user interaction at the frontend, and vice-versa. We discuss the synergistic design behind both the frontend user experience and the backend architecture of *DICE*; and, present a demonstration that allows the user to fluidly interact with billion-tuple datasets within sub-second interactive response times.

1. INTRODUCTION

Extracting insights from data for applications such as business intelligence have traditionally relied on batch-oriented materialization of multidimensional aggregates or full data cubes [7]. Lately, there been a powerful trend towards providing analytics in real-time or near-real-time. Prematerialization is extremely expensive from both time and space perspectives due to the exponential space of possible aggregations. The challenges are further exacerbated in the case of ad-hoc analysis using data cubes. Due to the size and dimensionality of the data, in spite of the availability of powerful distributed infrastructure, it may not be possible to aggregate all of the data within interactive response times.

Studies conducted in the Human-Computer Interaction domain [14, 21] have heavily motivated a sub-1000ms limit for query response time: the minimum time for the system to respond with a result without impeding the user's interaction "flow". Empirical tests on our system have shown the same, along with an lower limit of 5000ms for a user to peruse the query results.

We introduce the *DICE* frontend and backend, an integrated system built for easy traversal over a data cube. *DICE*, expanded as *"**D**istributed and **I**nteractive **C**ube **E**xploration"* is based on four core observations. First, we observe that ad-hoc data cube exploration is always performed in **sessions**, comprising multiple queries in succession, motivating our session-based query model, as opposed to dealing with queries as isolated one-off events. Such a user behavior is common in the case of exploration of data cubes where the user is trying to gain insights by inspecting various regions of the data cube at varying resolutions. Second, due to challenges of scale and expectations of query response time, it is necessary (and sufficient) to aggregate over a **sample** of the data. As shown in Figure 4, the user can manipulate the slider to conveniently run queries at different sampling rates, thereby allowing a user-specified trade-off between accuracy and response time. Third, aggregations can be performed in **parallel in a distributed environment**, further reducing response times. *DICE* achieves latencies that enable a fluid and interactive cube exploration experience, and provides the corresponding error bounds on aggregated results. Further, it provides the user with an intuitive easy-to-use interface that aids in the data cube exploration process. Fourth, as the user studies the query results, the backend is often idle – we have an opportunity to **speculatively execute** queries they might run next and cache them.

2. SYSTEM OVERVIEW

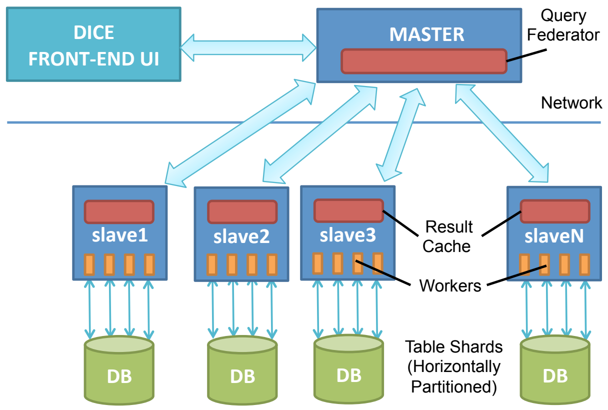


Figure 1: System Architecture

Figure 1 presents an overview of the *DICE* system, detailed in [11]. Our system employs a hierarchical master-slave architecture backend, connected to the frontend UI. All queries are issued to the master by the UI, and responded to by the master. The master in turn distributes these queries amongst the slave nodes. Multiple *table shards* are present at each slave node. The shards are atomic and read-only. The master maintains the catalog of the shards which aids in distribution of the workload. Upon an update or deletion of a shard, this information can be conveyed to the master, enabling the handling of rapidly changing data. An in-memory LRU result cache is maintained at the slaves. Previously executed queries can be answered using a simple lookup, reducing response times.

Given a user query, *DICE* enumerates possible speculative queries using the *faceted exploration model* (described in Section 3) and calculates their corresponding likelihood from the query logs. Due to the scale of the data and the number of speculative queries, it is not possible to execute all the speculative queries at the highest sampling rate. Hence, we choose a subset of queries to execute, and cache their results to maximize the probability of the next user query being present in the query cache.

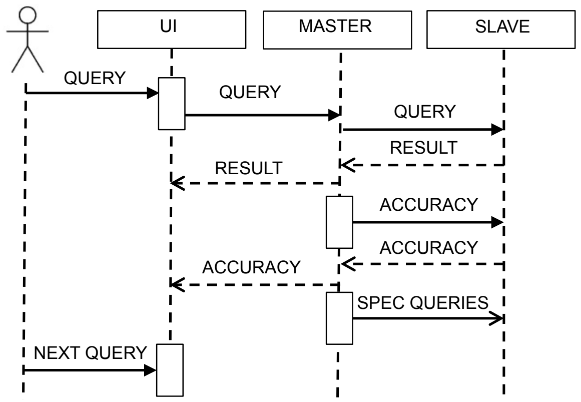


Figure 2: The user query is distributed by the master and executed on the slave nodes, followed by query for estimation of accuracy, followed by a set of speculative queries that populate the cache to reduce latency for the next query.

Query Flow: Figure 2 describes the overall query flow of *DICE* in terms of the sequence in which the user query, accuracy queries, and the speculative queries are run. The user query is run as soon as the user requests for it. After receiving the query result, the *accuracy* queries are run with the additional measures of *count*, *sum*, and *variance* which are needed to estimate the combined sampling accuracy for the measures *mean* and *sum*. Upon execution of the accuracy queries, the speculative queries are scheduled.

Speculative queries are run greedily: the order is based on a combination of the likelihood of the query occurring (derived using a query log), and the estimated gain in accuracy from that query result. Queries are run until the slave query caches are filled or the user enters their next query.

3. FACETED EXPLORATION MODEL

For any aggregation in the data cube, the number of possible follow-up queries is extremely large (number of groups in the cube - 1). This creates a challenge for our speculative execution technique – which queries do we pick? Thus, we present a model where the user explores the cube in a series of *faceted traversals*, thereby, restricting the space of queries. The *faceted cube exploration* model is used to guide user exploration at the frontend, and also to speculate and execute a subset of the next possible queries at different sampling rates so as to maximize the probability of the next query being present in the query cache at a high sampling rate. This synergistic model enables us to design an intuitive and easy to use interface to explore the data cube and, at the same time, reduce response time using speculative execution.

In the faceted exploration model, a facet of a cube region consists of a subset of groups obtained as a result of a query consisting of a `GROUP BY` on a single dimension and being bound on the remaining dimensions of the region, using a conjunction of `WHERE` clauses. The successive facet will be of type **parent** in the case of a rollup, **child** in the case of a drilldown, **sibling** in the case of change of a single dimension value and **pivot** for a change in the inspected dimension [11]. Prior work has used traversals such as *roll-up*, *drill-down*, and *pivot* in the context of exploration of data cube regions [7, 19, 20].

Faceted exploration builds upon this, providing a simple yet complete access to common query traversals over the data cube. Using faceted exploration, a user is able to *fully* explore the data cube – all cube groups can be explored using facets, and it is possible to traverse from any facet to another [11].

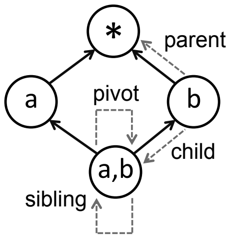


Figure 3: Faceted Cube Exploration Traversals

4. USER EXPERIENCE

4.1 System Design Principles:

We built the frontend and the backend of the *DICE* system based on the following design principles. Unlike typical systems, the query model assumes query sessions, interactivity, sampling, and speculation as key concepts.

Faceted Exploration: User interaction is strongly aligned with the faceted model. Exposing faceted traversals as UI elements has been shown to be effective in data cube traversal [11]. Faceted exploration helps inform the user about their location in the data cube and helps enumerate the neighboring facets which make up the set of speculative queries. While this aids the user in cube traversal, it also allows us to execute a subset of the speculative queries during the result perusal time, thus reducing the latency of the user query, and enhancing the synergy between user guidance and query latency reduction. For our backend, we draw inspiration from the HCI work on mixed initiative models [10], executing the most likely followup queries resulting into reduced expected latency for the user query.

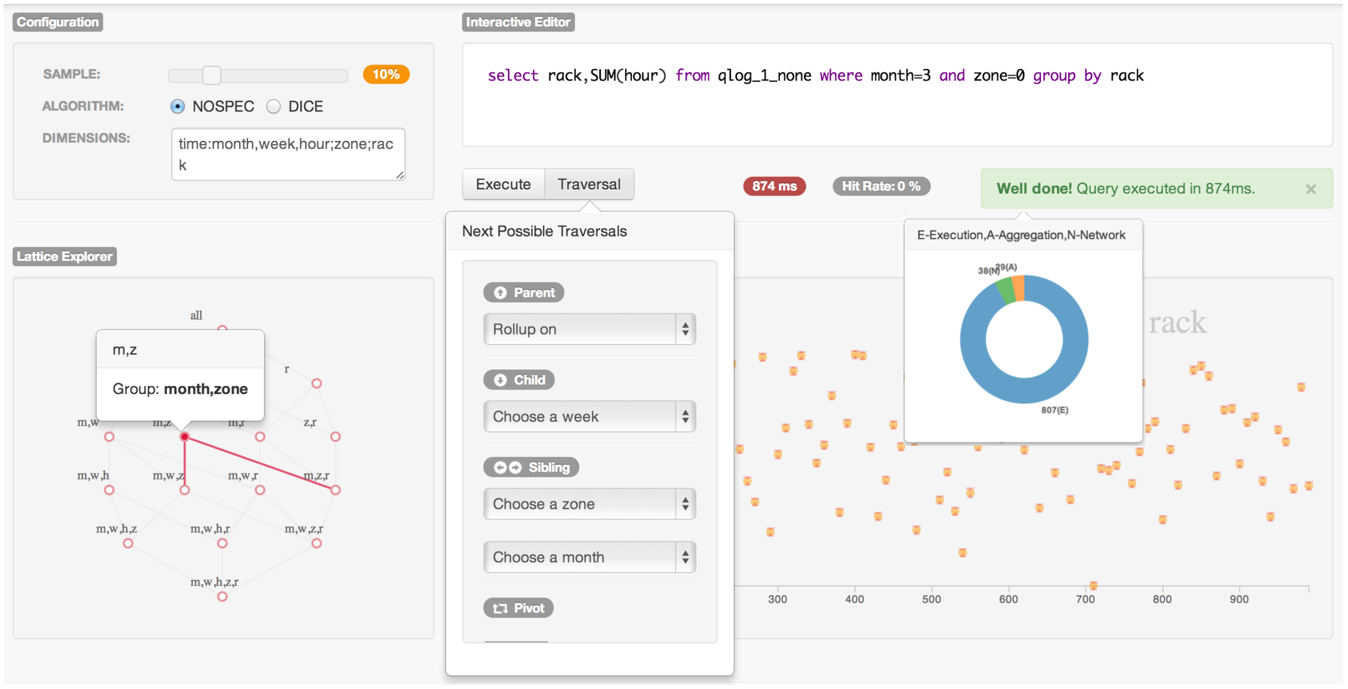
Latency vs Accuracy: Users should be able to interactively adjust sampling rates based on their needs, resulting in lower latency but higher errors. This is key for interactive cube exploration since depending on the number of sampled data points, skew of the data, selectivity of the *WHERE* predicate and the user desired accuracy, the rate of sampling should be allowed to vary, giving greater control over analysis to the user.

Results First: The results of a query should be prioritized and shown first, over the estimation of errors. Running the accuracy queries which consists of additional measures added to the user query results into an additional time expense of around 20%. As is evident from the query flow, we give greater importance to faster results compared to the accuracy estimation and hence, run the accuracy queries after the initial user query is executed. On the user interface, this is presented as large circles in the scatter plot, which later resolve into smaller points with error bars.

4.2 User Interface:

We now describe various parts of the *DICE* frontend user interface, as shown in Fig. 4. As described in the design principles, the query interface is inspired by the redefined sampling-aware query paradigm and the speculative execution capabilities of the backend.

Configuration: In the top left corner of the UI, a user can specify the sampling rate, the choice of whether to use speculative execution (Algorithm: DICE) or not (NOSPEC).



SAMPLE: 10%

ALGORITHM: [✓ NOSPEC] DICE

DIMENSIONS: time:month,week,hour;zone;rack

```
select rack,SUM(hour) from qlog_1_none where month=3 and zone=0 group by rack
```

874 ms

Hit Rate: 0 %

Well done! Query executed in 874ms.

m,z

Group: **month,zone**

Figure 4: ***DICE*** **User Interface that allows for interactive, intuitive cube exploration.**

Lattice Explorer: As the user inputs the dimensions and schema, the hierarchy-aware lattice is computed on the fly and displayed, allowing the user to get a feel for the data cube. Hovering over a region highlights the possible faceted traversals in the lattice, surfacing followup queries that are likely to be speculated.

Interactive Query Editor: One way to use the query editor is simply to enter the SQL query in the text box. Another more intuitive, easy-to-use option is provided below it in the form of the **Traversal** button. Clicking on it provides options to add a dimension resulting in a *child* traversal; remove a *WHERE* predicate causing a *parent* traversal; change the value of a *WHERE* predicate, thereby, inducing a *sibling* traversal; or swap a dimension in the case of a *pivot* traversal. In the case that the *WHERE* predicate is to be changed, the system will prompt the user with the range of values for the dimension. This allows the user to edit the query without advanced knowledge of the query language or schema. After the changes have been made to the SQL query, the query is automatically executed. Thus, the *Interactive Query Editor* allows the user to perform a *full* faceted cube exploration by providing different traversal options in a visual manner without having to type any part of the SQL query, enabling the user to navigate the entire data cube in a series of clicks. It is up to the user to use the more visual nature of the *Interactive Query Editor* or participate in a more hands-on fashion by entering the query themselves.

Execute: Below the editor is an *Execute* button which will need to be clicked in case the query is manually entered into the *Query Editor*. In case the interactive features of the *Editor* are used, there is no need for the **Execute** button to be clicked.

Timing Statistics: As a diagnostic aid, *DICE* also shows the response time, latency, cache hit rate, and a breakdown of the execution time between the backend execution, network transfer, and result aggregation. This helps provide feedback to the user about split up of the execution costs as well as how well the system is able to model user behavior.

Histogram Viewer: Result of the aggregation are presented at the bottom. The viewer displays a scatter plot as soon as possible to the user, maximizing utility. As discussed in Section 2, since queries are typically performed over sampled data, the system then issues a secondary set of queries (accuracy queries) to infer the error estimates, which it overlays as the next step, by adding error bars. The user views the result and then performs the next set of actions as part of the cube traversal.

5. DEMONSTRATION

During the demonstration, we will be running *DICE* on a distributed system in a single master, multiple slave configuration. The generated dataset will comprise of **1 billion rows** with 7 columns of type long, sharded uniformly across all nodes with a default table shard size of 1M rows. The data consists of cardinalities ranging from tens to a few thousands. We have shown [11] that *DICE* scales up to a billion tuples providing sub-second latencies using an Amazon $EC2$ configuration of 1 master and 50 slaves of the `c1.xlarge` type. The goal of the set-up is not to emphasize the size of the dataset but the distributed nature of the system necessary for scaling out, the benefits of speculative caching, and the ease in usability of the system at scale.

Scenario: Interactive cube exploration has applications in a a variety of fields, including business intelligence and data-driven science. Another popular scenario for interactive cube exploration is the management of cloud infrastructure. Tools such as VScope [23] have been developed for datacenter monitoring. For each setup, a handful of operations personnel manage tens of thousands of nodes, each with multiple virtual machines. Each instance produces a plethora of events which are logged to track performance, detect failures, and investigate systems issues. Each event item can be understood as a tuple with several fields, and each analytical task can be considered as the exploration of a cube over the logged data. Queries are ad-hoc, and due to the time-critical nature of the task, a system that allows for fast, interactive aggregations are highly desirable. During our demonstration we will walk through this use case with a query session. The queries will be logged and their execution will be repeated without speculation to demonstrate the difference. A query session including the following queries will be executed:

```
SELECT rack, AVG(iops) FROM events
WHERE datacenter = "EU" AND hour = 6 GROUP BY rack;
```

Such a query can be used to identify problematic I/O rates across *racks* which could cause failures in a datacenter over *time*. Next, the user changes the value of *hour* to *7* resulting in a *sibling* query. Then, the user wants to `GROUP BY` on hour and choose a particular *rack* to investigate how it is behaving over time which would equate to a *pivot* traversal:

```
SELECT hour, AVG(iops) FROM events
WHERE datacenter = "EU" AND rack = "r1"
GROUP BY hour;
```

Next, they wish to generalize on *rack* to get an idea of how well a datacenter is behaving over time, resulting in a *parent* query:

```
SELECT hour, AVG(iops) FROM events
WHERE datacenter = "EU" GROUP BY hour;
```

They then choose to generalize on the *location* dimension which would run a *parent* query. Following that, they choose another *datacenter*, giving a *child* query:

```
SELECT hour, AVG(iops) FROM events
WHERE datacenter = "ASIA" GROUP BY hour;
```

Above set of queries form a query session that the operations personnel might use to identify the problematic I/O rates across different racks spread across multiple data centers over time.

6. USER STUDY

We studied the impact of response times using a blind "taste test". Users were asked to explore the cube using the faceted model over 10 queries: they were not told if the speculation was turned on or off. After the session, the same session was repeated in the other mode (speculation was turned on if it off before, and vice-versa). For each user, we recorded time taken for the session and asked them about user satisfaction. The results were unanimous: Every user was able to discern the speedup and preferred DICE, and users saved an average of 7s over the mean session time of 54s.

7. RELATED WORK

Numerous business intelligence tools [2, 6, 12, 24, 8] provide visualization and interactive interfaces to large multidimensional datasets. Ad-hoc analysis over large datasets has been made popular with the availability of languages such as SCOPE [5], Pig [16] and Hive [22], which translate to MapReduce-oriented flows that are not ideal for interactive workloads. Work by Sarawagi et al. in the mining of *interesting* cube regions [19], exposing operators that further enable exploration [20] and that of Rizzi et al. in query personalization [18] are complementary to the ad-hoc exploration our work focuses on. Cetintemel et al. [4] provide a vision for a system to guide the user in interactive querying. Olston et al. [15] propose means for interactive analysis of web-scale data with the help of query templates. Techniques in online aggregation [9] have similar motivations to ours, but do not consider the influence of the interaction on system design. BlinkDB [1] considers only single queries (as opposed to query sessions) and computes an offline sample of numerous column combinations at multiple resolutions. *DICE*, on the other hand, uses an online sampling approach to get the query results and accuracy specifically targeted towards faceted cube exploration and as such is complementary to it.

8. CONCLUSION AND FUTURE WORK

We introduced the idea of simultaneously designing interaction and database architecture in *DICE*, a distributed, interactive cube exploration system. *DICE* provides a natural and intuitive way of faceted exploration of data cubes. The visually enhanced and easy point-and-click interface provided by *DICE*'s frontend helps not only novice users but also expert users by guiding them through the complexities of the data cube in a faceted exploration model. While typical data warehouses are designed around improving latencies for the user query, using the fact that in OLAP scenarios, queries usually don't exist in isolation but as part of sessions, we can provide improved latencies by using speculative execution.

9. ACKNOWLEDGEMENTS

We thank Amazon, NEC Labs America, and Yahoo! Labs for supporting parts of this work.

10. REFERENCES

[1] S. Agarwal, A. Panda, S. Madden, B. Mozafari, et al. Blink and It's Done: Interactive Queries on Very Large Data. *VLDB*, 2012.
[2] A. Buja, D. Cook, and D. F. Swayne. Interactive High-Dimensional Data Visualization. *Computational and Graphical Statistics*, 1996.
[3] Y. Cao, C. Chen, F. Guo, D. Jiang, et al. Es 2: A cloud data storage system for supporting both oltp and olap. *ICDE*, 2011.
[4] U. Çetintemel, M. Cherniack, J. DeBrabant, Y. Diao, et al. Query Steering for Interactive Data Exploration. *CIDR*, 2013.
[5] R. Chaiken, B. Jenkins, P. Larson, B. Ramsey, et al. SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets. *VLDB*, 2008.
[6] A. Dubrawski, M. Sabhnani, et al. Interactive Manipulation, Visualization Analysis of Large Sets of Multidimensional Time Series in Health Informatics. *INFORMS*, 2008.
[7] J. Gray et al. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. *Data Mining and Knowledge Discovery*, 1997.
[8] P. Hanrahan. VizQL: A Language for Query, Analysis and Visualization. *SIGMOD*, 2006.
[9] J. Hellerstein et al. Online Aggregation. *SIGMOD*, 1997.
[10] E. Horvitz. Principles of mixed-initiative user interfaces. *CHI*, 1999.
[11] N. Kamat, P. Jayachandran, K. Tunga, and A. Nandi. Distributed Interactive Cube Exploration. *ICDE*, 2014.
[12] D. Keim, F. Mansmann, et al. Visual Analytics: Scope and Challenges. *Visual Data Mining*, 2008.
[13] A. Kemper et al. HyPer: A Hybrid OLTP&OLAP Main Memory Database System. *ICDE*, 2011.
[14] R. Miller. Response Time in Man-Computer Conversational Transactions. *FJCC*, 1968.
[15] C. Olston, E. Bortnikov, K. Elmeleegy, F. Junqueira, and B. Reed. Interactive Analysis of Web-Scale Data. *CIDR*, 2009.
[16] C. Olston, B. Reed, et al. Pig Latin: A Not-So-Foreign Language for Data Processing. *SIGMOD*, 2008.
[17] H. Plattner. A common database approach for oltp and olap using an in-memory column database. *SIGMOD*, 2009.
[18] S. Rizzi. New frontiers in business intelligence: distribution and personalization. *Springer-Verlag*, 2011.
[19] S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-Driven Exploration of OLAP Data Cubes. *EDBT*, 1998.
[20] S. Sarawagi and G. Sathe. i3: Intelligent, Interactive Investigation of OLAP Data Cubes. *SIGMOD*, 2000.
[21] B. Shneiderman. Response Time and Display Rate in Human Performance with Computers. *CSUR*, 1984.
[22] A. Thusoo, J. Sarma, N. Jain, et al. Hive-A Petabyte Scale Data Warehouse using Hadoop. *ICDE*, 2010.
[23] C. Wang et al. Vscope: middleware for troubleshooting time-sensitive data center applications. *Middleware*, 2012.
[24] J. Yi et al. Toward a Deeper Understanding of the Role of Interaction in Information Visualization. *VCG*, 2007.