Visualization of HashStash with Qt

Zhe Zhao

Department of Computer Science
Brown University
Providence, RI
zhe_zhao at brown.edu

May 8, 2015

Abstract

HashStash is a new abstraction for the multi-query optimization. By managing internal data structures which is used by the individual operators between query sessions, HashStash provides a new form to share the intermediate result. The benefit of using HashStash than current approaches for multi-query optimization is that current approaches ignore the dominance of the main memory system performance issues, such as code and cache efficiency. Most of the query processing system should be run using command in terminal, however this is not user friendly and lack of interactivity. In this case, we use Qt to build an interactive interface for HashStash to make it much easier to run and display visualized result.

1. Introduction

Today researchers , businesses and scientists collect and analyze more and more data in data warehouses. With the increasing amount of data, the number of users and applications querying data grows exponentially (Psaroudakis, Athanassoulis, Olma, Ailamaki 22). In this condition, big data framework has played a big role. To increase larger data set, data scientists can weight their analytical tasks by using big data framework. However a big challenge in query execution is the increasing concurrency, queries may share the similar work or touch the common data. So we introduce a new abstraction which is HashStash. By using HashStash, user can reuse and share data stractors between related queries much easier.

For most query processing system, user should run the system using command. It is lake of interactivity and not user friendly. Running queries in the back-end might be easy for the user who knows the system well, but it is too complicated for the user who is new to the HashStash system. Instead of complicated use, the second drawback of running a system using command in the back-end is that it has low fault tolerance. When user has a little mistake in the command, the system can not be run and user needs to re-do again. Interactive HashStash uses HashStash as its back-end system and Qt based front-end as the user interface. Instead of typing long command in the back-end, user can simply type number or use clickable box in the front-end to run the system and display visualized result.

2. Overview

Qt is a cross-platform application framework(“Qt (Software)”, wikipedia.com). It is used mainly for developing application software with graphical user interfaces (GUIs). It is cross-platform and it uses the

system's resources to draw windows, controls, etc so the application will get a native look (e.g on a Mac your app window will be lacking the menu bar and the menu bar will appear on the system's menu bar as it is the standard behavior on the Mac platform).

Since the back-end of HashStash is written by using C++. Qt which uses standard C++ gives us great control and stable output and also possible to work with libraries.

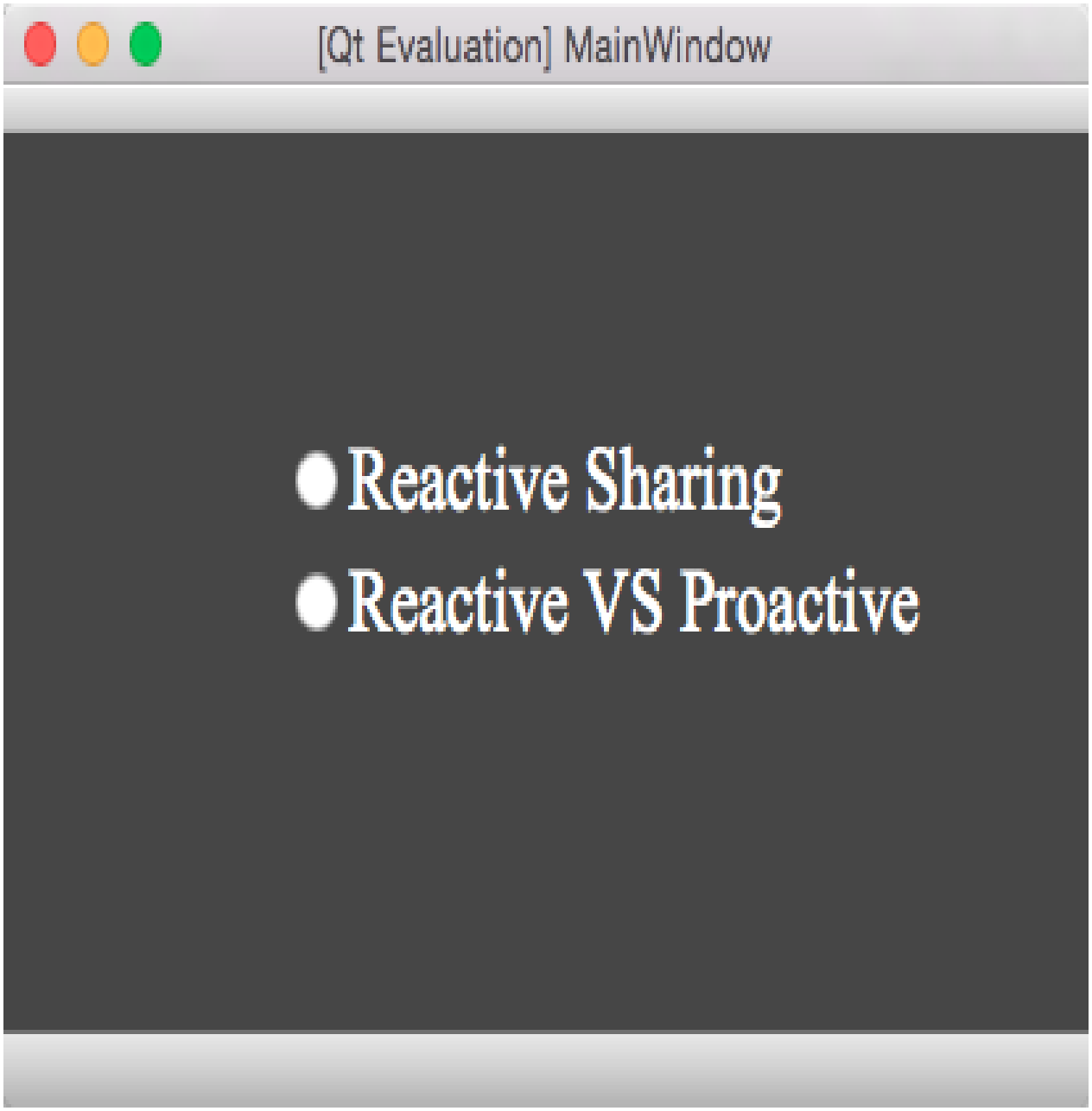
Another reason why we choose Qt is we can write html and CSS code in Qt to decorate our front-end.

3. Design & Implement

Queries sharing can be reactive to the essence existing sharing opportunity. By redesigning the query operators to maximize sharing opportunities, queries sharing can also be proactive(Psaroudakis, Athanassoulis, Olma, Ailamaki 22).

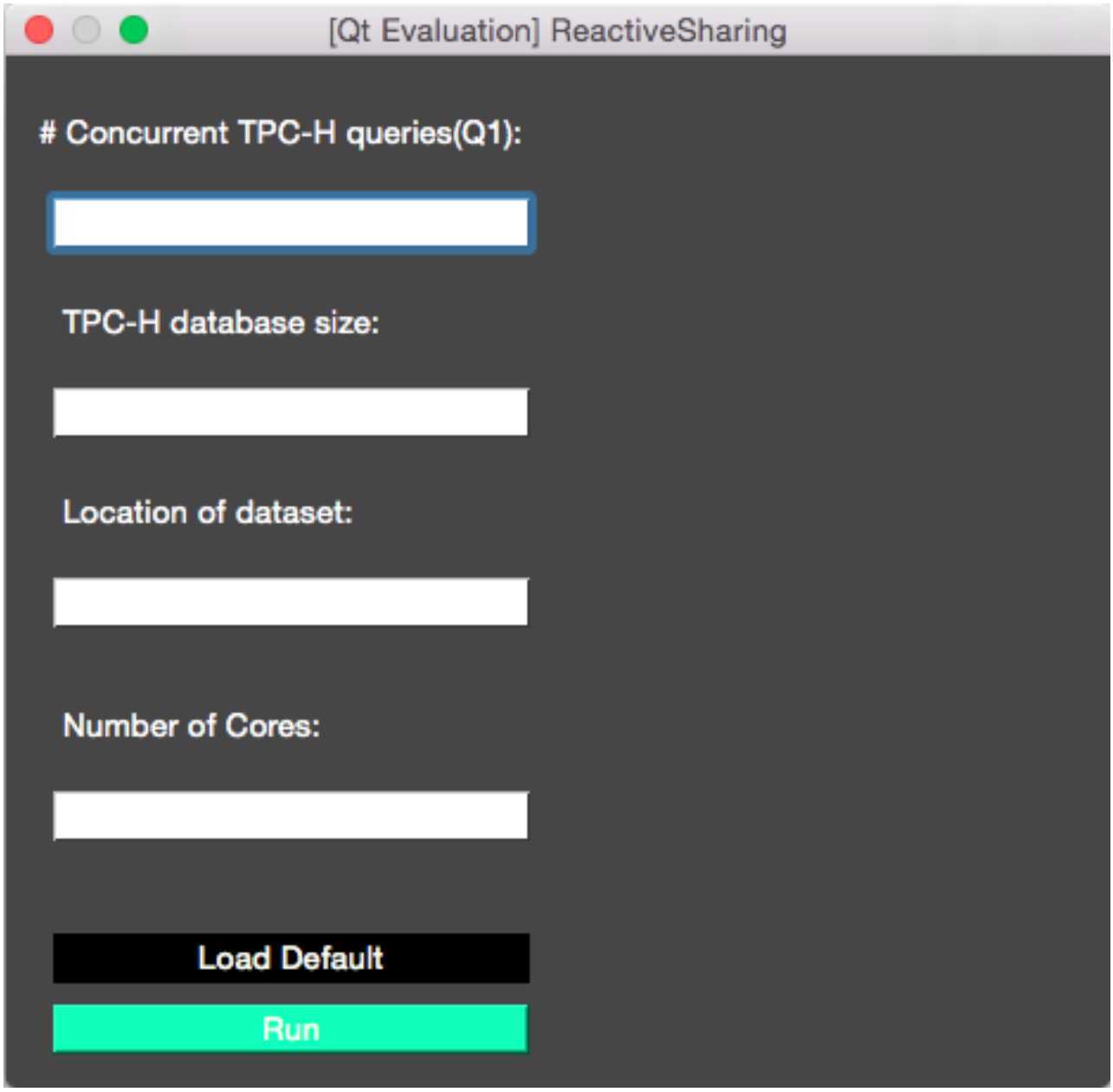
Based on these two properties, we make the interactive HashStash front-end into two parts, Reactive Sharing and Reactive VS Proactive. Reactive sharing shares common sub-plans and Global Query Plans' intermediate result. Proactive sharing evaluate a single query plan with shared operators*(*Psaroudakis, Athanassoulis, Olma, Ailamaki 22). We have three pages for the interactive HashStash front-end:

- Main Page
- Reactive Sharing
- Reactive VS Proactive



Main Page

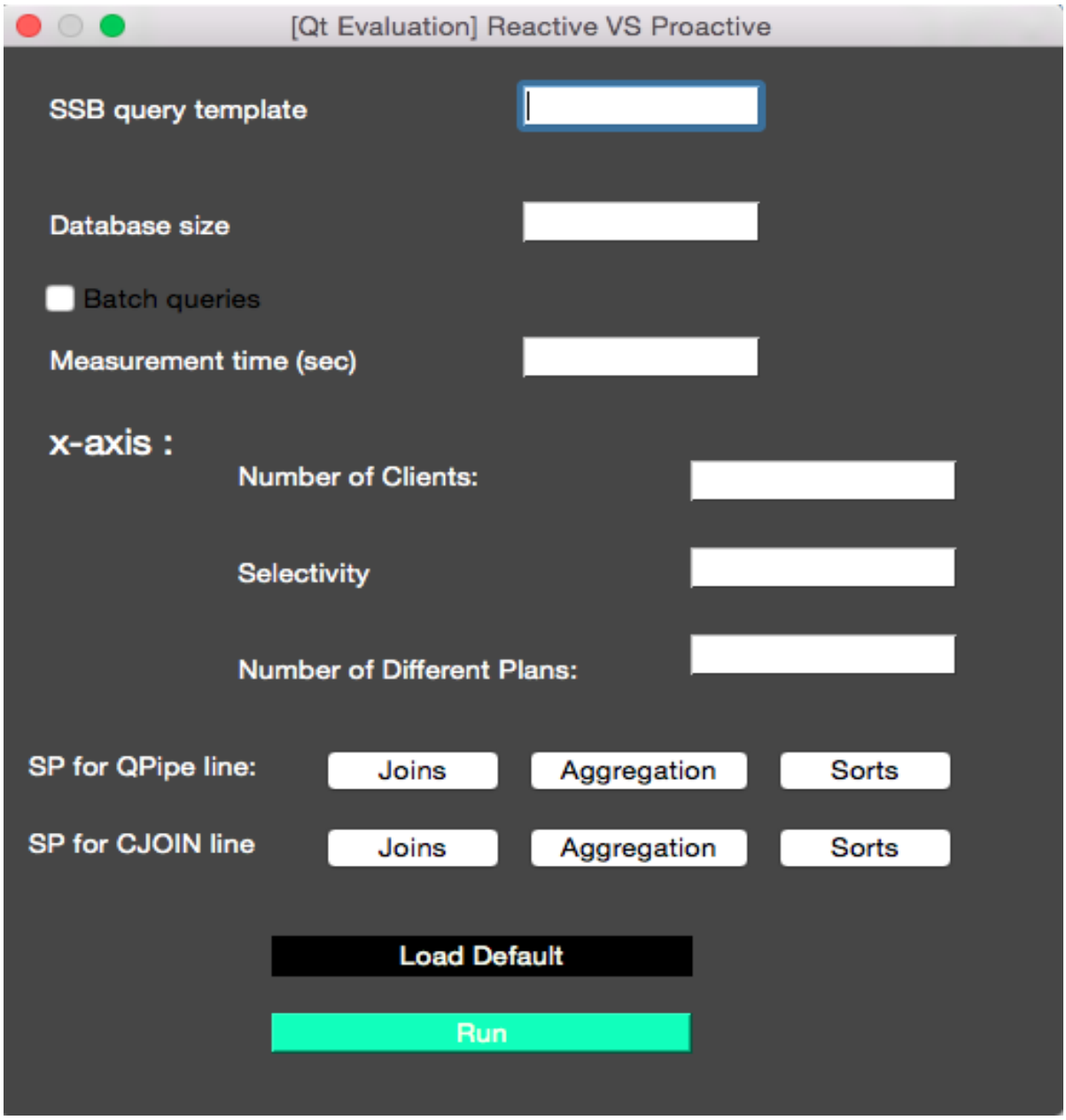
When user clicks “Reactive Sharing” button, it will jump to the second page “ Reactive Sharing” page.



Reactive Sharing page

User can enter the concurrent TPC-H queries, the size of TPC-H, location of database and the number of cores in the text area individually or click the default button to load the default value automatically. We show that pull-based sharing for Simultaneous Pipelining eliminates the serialization point imposed by the original push-based approach.

When click “Reactive VS proactive”, it will jump to the third page “ Reactive VS Proactive” page.



Reactive VS Proactive

We compare the performance of Simultaneous Pipelining and Global Query Plans through a sensitivity analysis. We also show that by mixing a query with common sub-plans, Simultaneous Pipelining can improve the performance of Global Query Plans.

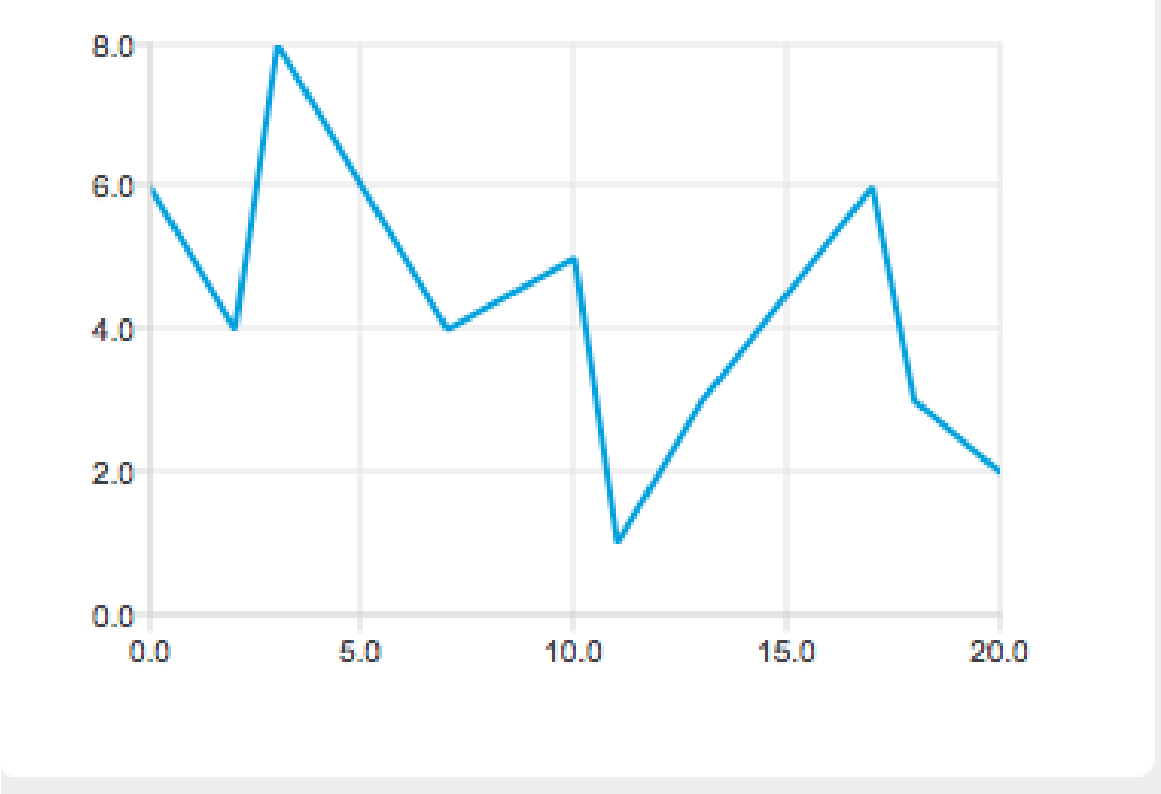
4. Visualization

In order to make the result more visualized and easy to read, we use two ways to display the result:

- Line Chart
- Bar Chart

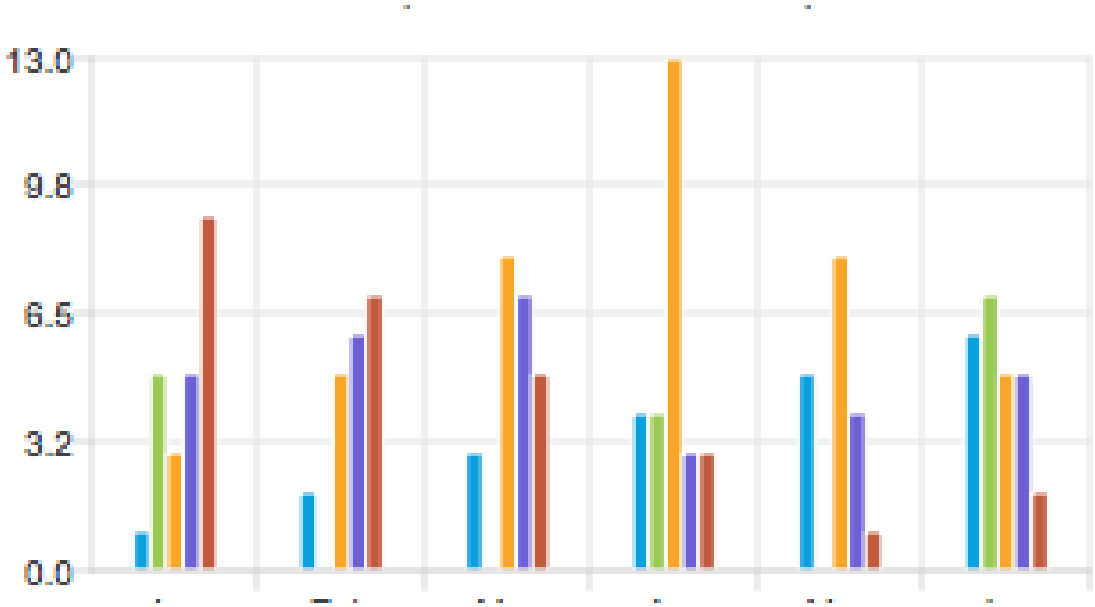
To create a line chart, a QVector instance is needed:

```
QVector<double> x(size), y(size);
```



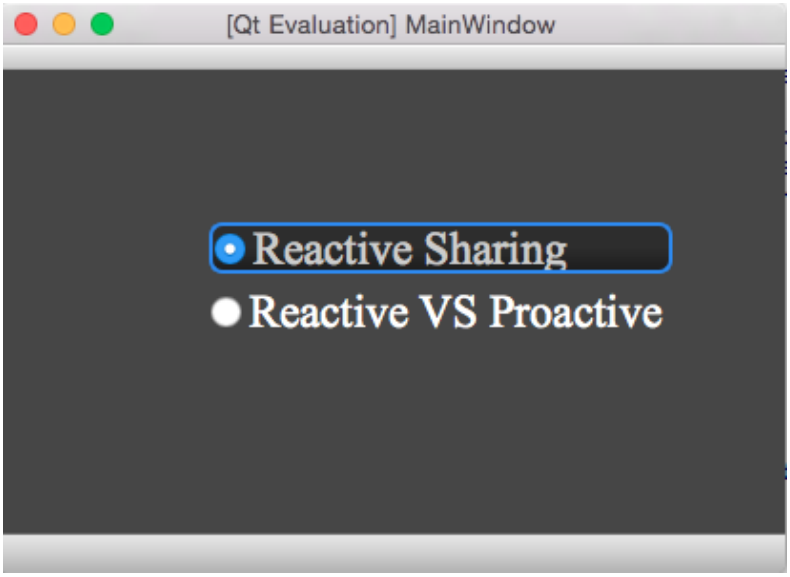
To create a bar chart, a QCPBars instance is needed:

```
QCPBars *myBars = new QCPBars(ui->widget->xAxis, ui->widget->yAxis);
```

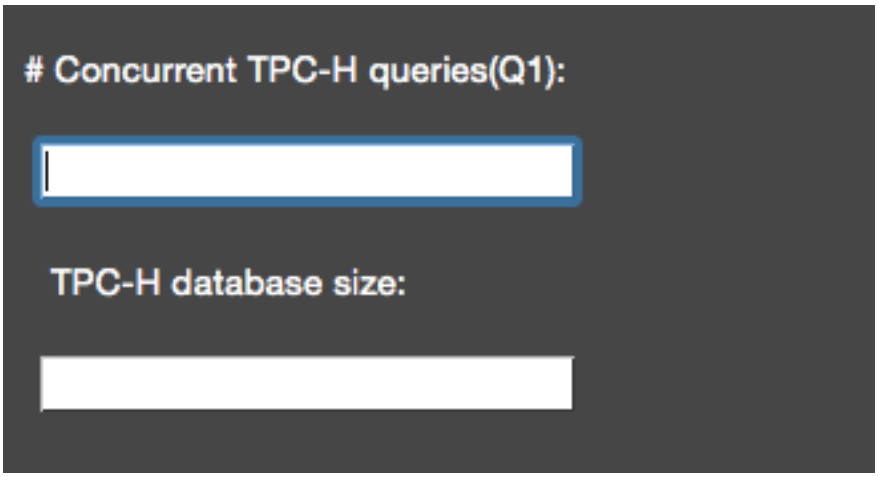


5. Other Features

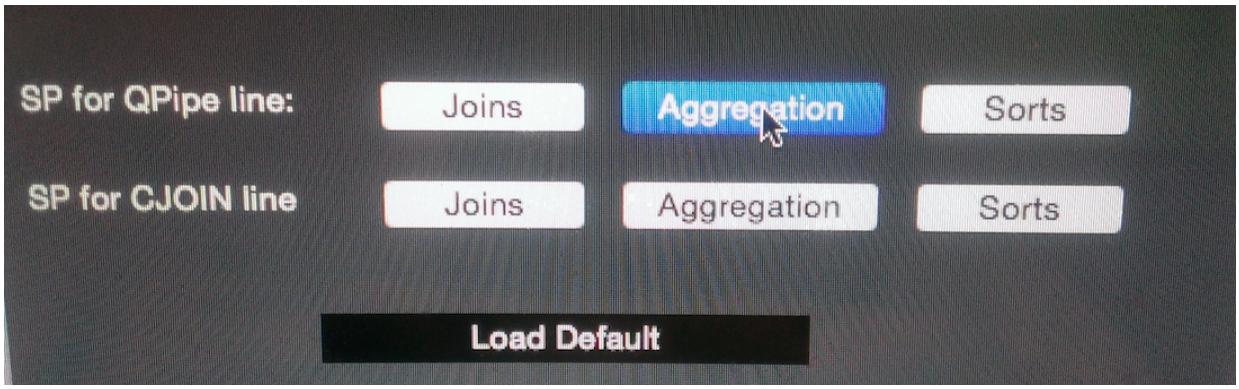
- Button Hover: when user puts mouse over the button, the hover selector will select the button.



- TextLine Highlight: when user chooses a textline and types the input, a highlight will appear to show which textline is using.



- Button Highlight: when user clicks a button, the button will be highlighted to show which button is clicked.



6. Summary

Since for most query processing system, user needs to run the system by using command in terminal which is completed, low fault tolerance and not user friendly, so this project build the front-end for HashStash by using Qt which can make HashStash system interactively. Qt is a cross-platform and it uses the system's resources to draw windows, controls, etc so the application will get a native look. It is C++ based and can combine html and CSS to decorate the page.

7. Acknowledgements

I would like to specially thank to my advisor Ugur Cetintemel who offered this project to me and gave me a chance to take part in the HashStash project. I would also like to thank to Kayhan Dursun who helped me get familiar with HashStash system by sending me related paper and communicated and discussed with me about the design and some key points of this project during the whole process.

Citation

1. I. Psaroudakis, M. Athanassoulis, M. Olma and A. Ailamaki. Reactive and Proactive Sharing Across Concurrent Analytical Queries. 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD 2014), Snowbird, UT, USA, June 22-27, 2014.

2. “Qt (Software).” *Wikipedia*. Wikimedia Foundation, n.d Web. 13 May 2015