

# **CLIENT-CENTRIC MOBILE OLAP**

By

**Zheng Xu**

B.Sc., Simon Fraser University, 2010

Project Submitted in Partial Fulfillment  
of the Requirements for the Degree of  
Master of Science

In the

School of Computing Science

Faculty of Applied Science

© **Zheng Xu 2012**

**SIMON FRASER UNIVERSITY**

**Summer 2012**

All rights reserved.

However, in accordance with the *Copyright Act of Canada*, this work may be reproduced, without authorization, under the conditions for "Fair Dealing." Therefore, limited reproduction of this work for the purposes of private study, research, criticism, review and news reporting is likely to be in accordance with the law, particularly if cited appropriately.

# Approval

**Name:** Zheng Xu  
**Degree:** Master of Science  
**Title of Project:** *Client-Centric Mobile OLAP*

**Examining Committee:**  
**Chair:** Ke Wang, Professor

---

**Wo-Shun Luk**  
Senior Supervisor  
Professor

---

**Jian Pei**  
Supervisor  
Professor

---

**Stephen Petschulat**  
External Examiner  
Senior Architect  
SAP Research

**Date Defended/Approved:** August 24, 2012

---

## Partial Copyright Licence



The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection (currently available to the public at the "Institutional Repository" link of the SFU Library website ([www.lib.sfu.ca](http://www.lib.sfu.ca)) at <http://summit/sfu.ca> and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author's written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

While licensing SFU to permit the above uses, the author retains copyright in the thesis, project or extended essays, including the right to change the work for subsequent purposes, including editing and publishing the work in whole or in part, and licensing other parties, as the author may desire.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library  
Burnaby, British Columbia, Canada

revised Fall 2011

## **Abstract**

As the architecture for clients of a client-server OLAP system, the client-centric system distinguishes itself from other clients in the marketplace by incorporating a light-weight data engine, such that the client is capable of answering queries offline when the related data are cached in the memory. In a previous study of client-centric OLAP system, a web-based client is built. Three different data visualization scenarios have been identified for which the client can pre-load data into the client as a sub-cube from the server, so that the user may drill-down the sub-cube, with the help of the data engine. In this research, the app-based version of the client-centric OLAP is implemented. It runs on an android tablet featuring a multi-core CPU. From the comparative performance data, it is shown that the app-based client is superior to the web-based client in performance in most computational tasks of the three data visualization scenarios, especially when a parallel version of the app-based OLAP is deployed.

**Keywords:** Client-Centric OLAP; Mobile OLAP; Data Visualization; Parallel Computing

## **Acknowledgements**

I express deep gratitude to my senior supervisor Dr. Wo-Shun Luk for accepting me as his student and for providing his guidance. This project would not have been possible without his knowledge and support. I also thank Dr. Jian Pei, Dr. Ke Wang, and Stephen Petschulat from SAP for serving on my defense committee. Lastly, I thank Johnny Zhang and Tim Hsiao for helping me to setup the experimental environment of this project.

# Table of Contents

|   |           |
|---|-----------|
| Approval .....  | ii        |
| Partial Copyright Licence .....                                 | iii       |
| Abstract.....   | iv        |
| Acknowledgements.....   | v         |
| Table of Contents .....   | vi        |
| List of Tables .....  | viii      |
| List of Figures.....  | ix        |
| <br>  |           |
| <b>1. INTRODUCTION.....</b>                                     | <b>1</b>  |
| 1.1 Platforms for OLAP Clients.....                             | 1         |
| 1.2 Client-Centric OLAP .....                                   | 2         |
| 1.3 Research Scope and Objectives.....                          | 3         |
| 1.4 Organization of the Report .....                            | 4         |
| <br>  |           |
| <b>2. SERVER IMPLEMENTATION .....</b>                           | <b>5</b>  |
| 2.1 OLAP Server .....   | 6         |
| 2.2 ADOMD .....   | 7         |
| 2.2.1 Metadata.....   | 9         |
| 2.2.2 Cell Ordinals .....                                       | 9         |
| 2.3 SOAP Web Service .....                                      | 12        |
| 2.4 Application Server.....                                     | 12        |
| <br>  |           |
| <b>3. MOBILE CLIENT IMPLEMENTATION.....</b>                     | <b>13</b> |
| 3.1 Mobile Clients in the Client-Centric Architecture .....     | 13        |
| 3.2 Mobile Thin Client in the Server-Centric Architecture ..... | 16        |
| 3.3 OLAP Data Engine.....                                       | 17        |
| 3.3.1 Cross Tabulation.....                                     | 17        |
| 3.3.2 Parallel Cross Tabulation .....                           | 21        |
| 3.4 OLAP Visualization Engine.....                              | 23        |
| 3.4.1 Data Grid Implementation.....                             | 23        |
| 3.4.2 Top-K Percentage Contribution .....                       | 24        |
| 3.4.3 Moving Average .....                                      | 26        |
| 3.4.4 Scatter Plot.....   | 27        |
| <br>  |           |
| <b>4. PERFORMANCE EVALUATION.....</b>                           | <b>30</b> |
| 4.1 Client-Centric Performance .....                            | 30        |
| 4.1.1 Metadata Download and Parsing .....                       | 30        |
| 4.1.2 Sub-Cube Download and Parsing.....                        | 32        |
| 4.1.3 Cross Tabulation Evaluation.....                          | 33        |
| 4.1.4 Parallel Cross-Tab Evaluation .....                       | 35        |
| 4.1.5 Data Grid Evaluation.....                                 | 38        |
| 4.1.6 Top-k Percentage Contribution Evaluation .....            | 39        |
| 4.1.7 Moving Average Evaluation.....                            | 42        |
| 4.1.8 Scatter Plot Evaluation .....                             | 43        |
| 4.2 Server-Centric Performance.....                             | 48        |

|  |           |
|--|-----------|
| 4.2.1 Top-k Comparison with Client-Centric Architecture.....           | 48        |
| 4.2.2 Moving Average Comparison with Client-Centric Architecture ..... | 50        |
| <b>5. CONCLUSION AND FUTURE WORK.....</b>                              | <b>52</b> |
| 5.1 Conclusion .....   | 52        |
| 5.2 Future Work .....  | 53        |
| <b>REFERENCES .....</b>  | <b>54</b> |

## List of Tables

|   |    |
|---|----|
| Table 1. MDX query result.....                                  | 7  |
| Table 2. Metadata information.....                              | 31 |
| Table 3. Sub-cubes information.....                             | 33 |
| Table 4. Top-k running time break down.....                     | 41 |
| Table 5. Sub-Cube information for scatter plot aggregation..... | 46 |



## List of Figures

|   |    |
|---|----|
| Figure 1. Client-centric OLAP architecture.....                                   | 5  |
| Figure 2. Server-centric OLAP architecture .....                                  | 5  |
| Figure 3. A three dimensional data sube.....                                      | 6  |
| Figure 4. MDX query .....   | 6  |
| Figure 5. XMLA Execute request.....   | 8  |
| Figure 6. XMLA Discover request .....   | 8  |
| Figure 7. Gender dimension and its hierarchy .....                                | 9  |
| Figure 8. MDX query for lowest level of aggregation .....                         | 9  |
| Figure 9. Fact table.....   | 10 |
| Figure 10. Hierarchy tree leaves selection.....                                   | 10 |
| Figure 11. Query result.....  | 11 |
| Figure 12. Cell table.....  | 11 |
| Figure 13. Android architecture overview.....                                     | 14 |
| Figure 14. App-based client screenshot.....                                       | 14 |
| Figure 15. Web-based client screenshot .....                                      | 15 |
| Figure 16. Fact table with the lowest level of aggregation .....                  | 17 |
| Figure 17. Cross-tab of Country and Education.....                                | 18 |
| Figure 18. Cross-tab algorithm.....   | 19 |
| Figure 19. Cross-tab of State-Province and Education.....                         | 20 |
| Figure 20. Metadata indexes.....  | 20 |
| Figure 21. Parallel cross-tab algorithm.....                                      | 22 |
| Figure 22. Data grid.....   | 24 |
| Figure 23. Top-k screen shot.....   | 25 |
| Figure 24. Moving average screen shot.....  | 27 |
| Figure 25. Scatter plot screen shot.....  | 28 |
| Figure 26. Cross-tab performance comparison.....                                  | 34 |
| Figure 27. Cross-tab speedup.....   | 35 |
| Figure 28. Cross-tab parallel performance.....                                    | 36 |
| Figure 29. Cross-tab parallel speedup.....  | 36 |
| Figure 30. Cross-tab parallel efficiency.....                                     | 37 |
| Figure 31. Web-based and parallel app-based cross-tab performance comparison..... | 38 |

|  |    |
|--|----|
| Figure 32. Parallel cross-tab speedup for app-based .....                  | 38 |
| Figure 33. Top-k performance comparison.....                               | 40 |
| Figure 34. Top-k speedup.....  | 40 |
| Figure 35. Moving average performance.....                                 | 42 |
| Figure 36. Moving average speedup.....                                     | 43 |
| Figure 37. Scatter plot rendering.....                                     | 44 |
| Figure 38. Scatter plot rendering speedup for web-based.....               | 44 |
| Figure 39. Large scatter plot rendering.....                               | 45 |
| Figure 40. Large scatter plot rendering speedup for web-based.....         | 45 |
| Figure 41. Scatter plot aggregation.....                                   | 47 |
| Figure 42. Scatter plot aggregation speedup for app-based.....             | 47 |
| Figure 43. Top-k server-centric compared with client-centric.....          | 49 |
| Figure 44. Top-k speedup for client-centric.....                           | 50 |
| Figure 45. Moving Average server-centric compared with client-centric..... | 51 |
| Figure 46. Moving Average speedup for client-centric.....                  | 51 |

# 1. INTRODUCTION

## 1.1 Platforms for OLAP Clients

In the marketplace, there is a large selection of OLAP (On-line Analytic Processing) client tools for the user to choose from. Traditionally, client tools run on a desktop connected to the server in a local area network. As web browsers became the user interface of choice, web-based client tools were introduced. Compared to desktop clients, a web-based client is more resource-constrained and consequently, performance and user experience suffer. On the other hand, it runs where internet access is available. There is also the benefit of user convenience, since there is nothing to install and maintain. The web-based client particularly appeals to casual users. In contrast, a desktop client nowadays can be as powerful as the machine the OLAP server runs on. It is capable of downloading a large dataset into its cache, and let the user analyze the data, and ask ‘what if’ questions, within his/her personal computing environment.

Over a last couple of years, a new application platform has emerged: mobile devices, e.g., smartphones and tablets. Smartphones and tablets are increasingly gaining popularity with the introduction of the iPhone, the iPad, and Google’s Android. It is estimated that 59% of mobile phone users have smartphones by the third quarter of 2011 [4]. In addition, tablets are ready to replace the traditional personal computer [5]. Today, a smartphone or tablet is often the only computing device a person uses today. However, tablets such as the iPad are not only attractive to consumers, but also to business users. Tablets are helping doctors, lawyers, and businessmen to do their work [10, 11]. It is

estimated that the mobile office software market in North America may reach \$6.85 billion by the year 2015 [12].

Unfortunately, these mobile devices are constrained in memory size, CPU power, and network bandwidth compared to laptops and desktop computers. Because of the above constraints, the mobile web browsers of smartphones and tables are very slow in opening web pages due to slow resource loading [3]. In addition, the authors in [6] demonstrated that the JavaScript execution time is 20 to 80 times longer on smartphones than on desktop computers.

Vendors of OLAP clients on mobile devices differ in their product strategies [2]. Some vendors simply move their web-based systems that are built for desktops onto mobile devices. Others develop mobile applications that run on the native operating system of mobile devices, such as Android. The arguments in favor of native mobile applications can be summarized into three points: fast performance, superior user experience, and offline access. On the other hand, web-based OLAP has its own advantages in terms of portability and security. Some of the leading vendors, such as SAP Business Objects [13] and IBM Cognos [36] offer their client tools on all three platforms: desktop, web-browser and native mobile OS, e.g., Android from Google and iOS from Apple.

## **1.2 Client-Centric OLAP**

In [27], Tim Hsiao developed a web-based OLAP prototype which incorporates a data engine for processing queries on the local data already downloaded from the server. He demonstrated the superiority of this approach by implementing three data visualization scenarios: top-K, moving average and scatter plot, where data may be pre-

loaded from the server. We will describe the first scenario to show how client-centric OLAP works.

Suppose a floor supervisor in an electronic store would like to get details about the top-k best sellers in the store in past few weeks in order to configure the floor display the next day. Usually, he would send a query to the server for top-k best sellers for a specific time, one at a time, by moving the slider on the user interface on his smartphone. With the data engine on the client side, a sub-cube with products and time dimensions can be downloaded to the client running on a browser for the drill-down and roll-up operations done entirely within the client. For this application, the user gets the same benefits due to offline data access, e.g., superior performance and enhanced user experience. To contrast these two approaches, we call them server-centric and client-centric OLAP respectively.

However, there exists little research on client-centric OLAP clients for mobile devices. Other researchers study data cube construction and compression [1, 14, 15, 16] and query recommendation [17]. One major issue of client-centric OLAP is security. A large amount of important data stored on mobile devices is a concern for companies that value privacy and security, and it is difficult to enforce data consistency. Another important reason is that the data cube is likely to be too massive for previous generations of mobile devices to store in memory. Furthermore, downloading the data cube to the mobile device can be costly and time consuming.

### **1.3 Research Scope and Objectives**

In this project, my focus is on client-centric mobile OLAP. In particular, I implement a client-centric OLAP prototype on a multi-core tablet running a version of

the Android operating system, hence an app-based client-centric mobile client. Being a client-centric system, this prototype is different from the mobile OLAP applications mentioned earlier because it contains a data engine for aggregation purposes. For comparison purpose, two versions of data engine are built: parallel and non-parallel versions.

The main objective of this project is to compare performance of the web-based OLAP, as described by Tim Hsiao, and the app-based client-centric OLAP. It is my hope that this study will increase the awareness of the client-centric OLAP systems in the OLAP industry. Should this type of Mobile OLAP system be of interest to a vendor, this study might help them to determine whether web-based or app-based system is more desirable for them.

Note that Tim's prototype has been significantly modified so that we can do an apple-to-apple comparison. In addition, Tim's prototype runs on a browser in a desktop system, and some of Tim's performance data are simulated. Thus this is the first time detailed experimental data are available for both web-based and app-based client-centric OLAP systems. Therefore, we include also the performance comparison with the server-centric OLAP system.

## **1.4 Organization of the Report**

This report has five chapters in total. The next chapter provides a detailed description of the server-side components of OLAP system architectures. Afterwards, the implementation of the mobile clients is explained in chapter 3. Then chapter 4 discusses the performance evaluation of the client-centric and server-centric architectures. Finally, chapter 5 concludes this report and discusses possible future work.

## 2. SERVER IMPLEMENTATION

OLAP systems have two architectures: client-centric and server centric. Figure 1 provides an overview of the client-centric architecture and figure 2 provides an overview of the server-centric architecture. There are four different components on the server-side. In this chapter, we explain each server-side component of both OLAP architectures.

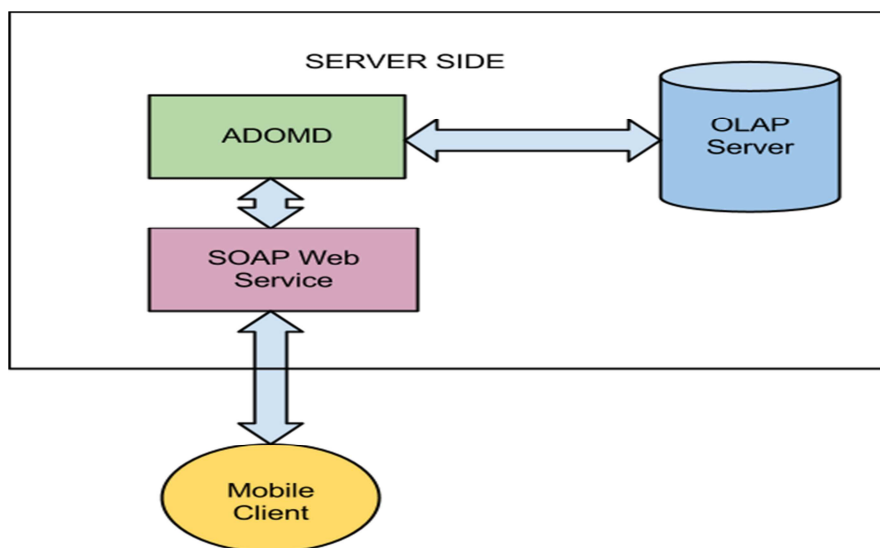


Figure 1. Client-centric OLAP architecture

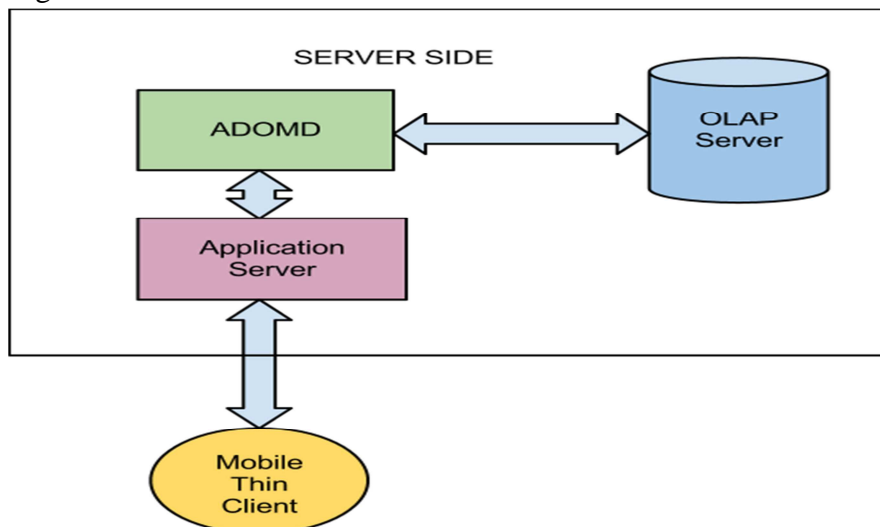


Figure 2. Server-centric OLAP architecture

## 2.1 OLAP Server

The foundation of any OLAP system is the OLAP cube. A cube is a set of numerical data categorized by multiple dimensions. This set of numerical data is the measures. For example, a two dimensional cube is a Cartesian product of the dimensions  $A$  and  $B$ , where  $A$  and  $B$  are both sets of items. Each element of the Cartesian product is assigned to a numerical value from a measure. When there are more than three dimensions, a cube is called a hypercube. A sub-cube is a subset of a cube. Figure 3 shows a three dimensional data cube with the dimensions Product, City, and Date.

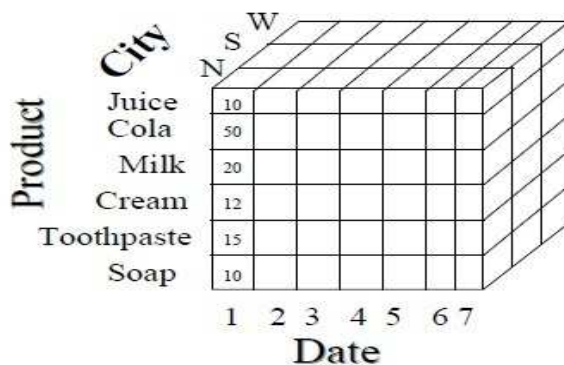


Figure 3. A three dimensional data cube [21]

The OLAP server we use is Microsoft Analysis Services 2008 R2. The OLAP server retrieves a sub-cube from the OLAP cube by using Multidimensional Expressions (MDX) [18], a query language for OLAP databases. An example of an MDX query is shown in figure 4.

```
SELECT
  {[Customer].[Education].Children} ON 0,
  {[Customer].[Gender].Children} ON 1
FROM [Adventure Works]
WHERE [Measures].[Internet Sales Amount]
```

Figure 4. MDX query



This sub-cube is defined by two dimensions: [Customer].[Education] and [Customer].[Gender]. The measure is Internet Sales Amount. The OLAP server processes this query and the result of this MDX query is displayed in table 1.

|        | Bachelors      | Graduate Degree | High School    | Partial College | Partial High School |
|--------|----------------|-----------------|----------------|-----------------|---------------------|
| Female | \$5,076,191.40 | \$2,819,687.51  | \$2,169,808.44 | \$3,904,283.72  | \$843,647.62        |
| Male   | \$4,823,951.36 | \$2,640,872.75  | \$2,468,217.63 | \$3,819,259.17  | \$792,757.64        |

Table 1. MDX query result

To allow application software to execute MDX queries, we use a standard protocol called XML for Analysis (abbreviated as XMLA).

## 2.2 ADOMD

ADOMD is a Microsoft .Net framework that implements the XML for Analysis (XMLA) protocol for communication with Microsoft Analysis Service. XMLA [19] is an Application Programming Interface (API) for data access in OLAP. It is based on XML, SOAP, and HTML. It allows an application to communicate with an OLAP server. XMLA has two methods [20]: Execute and Discover. The Execute message is used for sending query request to the OLAP server. The Execute request embeds an MDX query. The format of the result can be specified by the user. An example of an Execute request is shown in figure 5. The Discover request is used to retrieve information and metadata from the OLAP server. Like the Execute request, the format of the result of a Discover request can be also be specified by the user. Figure 6 shows an example that retrieves information about the cubes in an OLAP database. The results of both the Execute

message and the Discover message are highly detailed, containing information on dimensions and axes.

```
<Execute xmlns="urn:schemas-microsoft-com:xml-analysis">
  <Command>
    <Statement>
      SELECT
        {[Customer].[Education].Children} ON 0,
        {[Customer].[Gender].Children} ON 1
      FROM [Adventure Works]
      where [Measures].[Internet Sales Amount]
    </Statement>
  </Command>
  <Properties>
    <PropertyList>
      <Format>Multidimensional</Format>
      <AxisFormat>TupleFormat</AxisFormat>
    </PropertyList>
  </Properties>
</Execute>
```

Figure 5. XMLA Execute request

```
<Discover xmlns="urn:schemas-microsoft-com:xml-analysis">
  <RequestType>MDSHEMA_CUBES</RequestType>
  <Restrictions>
    <RestrictionList>
      <CATALOG_NAME>Adventure Works DW 2008R2</CATALOG_NAME>
    </RestrictionList>
  </Restrictions>
  <Properties>
    <PropertyList>
      <Catalog>Adventure Works DW 2008R2</Catalog>
      <Format>Tabular</Format>
    </PropertyList>
  </Properties>
</Discover>
```

Figure 6. XMLA Discover request

### 2.2.1 Metadata

Before retrieving the sub cube, the client first needs to request the SOAP web service for the metadata. The metadata contains a dimension's hierarchy. Figure 7 shows the hierarchy of the Gender dimension from the *Adventure Works* data cube. After selecting a dimension, detailed information on the dimension's hierarchy will be sent to the client. The information contains names of each node in every level of the hierarchical tree. The web service uses ADOMD to send XMLA Discover messages to retrieve the metadata from the OLAP server.

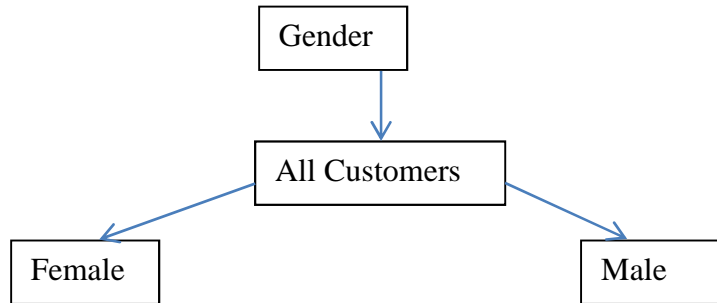


Figure 7. Gender dimension and its hierarchy

### 2.2.2 Cell Ordinals

After downloading the required metadata, XMLA is used to retrieve sub-cubes with the lowest level (the leaves of the hierarchy tree) of aggregation possible to perform aggregations such as cross-tab, drill-down and roll-up later by the client or the application server. The SOAP web service receives an MDX query (figure 8) created by the client.

```
SELECT {[Measures].[Internet Sales Amount]} on axis(0),  
DESCENDANTS([Customer].[Education].[All Customers],,LEAVES) on axis(1),  
DESCENDANTS([Geography].[Geography].[All Geographies],,LEAVES) on axis(2)  
FROM [Adventure Works]
```

Figure 8. MDX query for lowest level of aggregation

Since our goal is to minimize the amount of data sent over the network, we do not create a fact table (figure 9). Therefore we do not use ADOMD to execute this query (figure 8) because ADOMD produces a fact table from the results of the XMLA execute message.

|  | [Geography].[Geog | [Geography].[Geog | [Geography].[Geog | [Geography].[Geog Code].[MEMBER_C | [Customer].[Educat  | [Measures].[Interne Sales Amount] |
|--|-------------------|-------------------|-------------------|-----------------------------------|---------------------|-----------------------------------|
|  | Australia         | New South Wales   | Alexandria        | 2015                              | Bachelors           | 9900142.7571                      |
|  | Australia         | New South Wales   | Alexandria        | 2015                              | Graduate Degree     | 5460560.2513                      |
|  | Australia         | New South Wales   | Alexandria        | 2015                              | High School         | 4638026.0686                      |
|  | Australia         | New South Wales   | Alexandria        | 2015                              | Partial College     | 7723542.8848                      |
|  | Australia         | New South Wales   | Alexandria        | 2015                              | Partial High School | 1636405.2589                      |
|  | Australia         | New South Wales   | Coffs Harbour     | 2450                              | Bachelors           | 9900142.7571                      |
|  | Australia         | New South Wales   | Coffs Harbour     | 2450                              | Graduate Degree     | 5460560.2513                      |

Figure 9. Fact table

Instead we use XMLA directly to execute this MDX query (figure 8). XMLA executes this query by embedding it in an Execute message. This query returns the leaves of each dimension's hierarchy tree (Figure 10), which are represented by a list of cells.

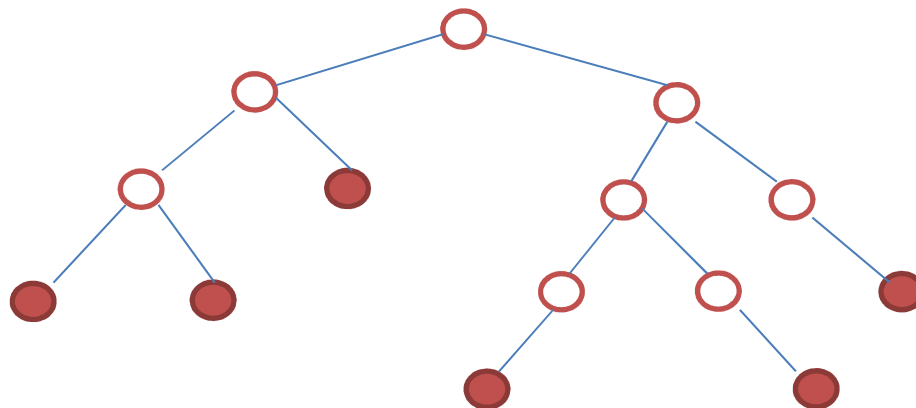


Figure 10. Hierarchy tree leaves selection

The result of the query is shown in figure 11. Each cell represents an element in the Cartesian product of the dimensions. Each cell is identified by an integer called *CellOrdinal* and is assigned to a numerical value from the measures.

```
<Cell CellOrdinal="0">
<Value xsi:type="xsd:double">9900142.7571</Value>
<FmtValue>$9900142.7571</FmtValue>
</Cell>
<Cell CellOrdinal="1">
<Value xsi:type="xsd:double">5460560.2513 </Value>
<FmtValue>$5460560.2513 </FmtValue>
</Cell>
```

Figure 11. Query result

Since the result of a XMLA query execution is highly detailed, we parse the result to retrieve only the numerical values such that the amount of data transmitted over the network is minimized. This list of numerical values becomes the sub-cube (figure 12) that will be sent to the client in the client-centric architecture. We call this list of numerical values the *cell table*.

```
9900142.7571
5460560.2513
4638026.0686
7723542.8848
1636405.2589
9900142.7571
5460560.2513
```

Figure 12. Cell table

## **2.3 SOAP Web Service**

The web service component serves as a middleware between the client and the ADOMD layer. This component is created using ASP .Net. It uses Simple Object Access Protocol (SOAP) as the protocol for the exchange of structured information. SOAP uses XML as a message format and uses HTTP for message transmission.

The web service has two major functions. It receives requests from the client, which could be requests for metadata, or a query in MDX. It processes the results from ADOMD and XMLA before sending them to the client in the client-centric architecture.

## **2.4 Application Server**

In the server-centric architecture, the application server is an extension of the web service component discussed in the last section. It receives query requests from the mobile thin client using SOAP as the message exchange protocol. However, it does not send the metadata and the sub-cube to the thin client. Instead, the application server uses the metadata and the sub-cube locally to compute OLAP operations such as cross-tab, top-k, and moving average before sending the results to the client.

## **3. MOBILE CLIENT IMPLEMENTATION**

In this chapter, we explain the implementations of the mobile client in the client-centric architecture and the mobile thin client in the server-centric architecture in sections 3.1 and 3.2, respectively. Then in section 3.3, we explain the implementation of the OLAP data engine. Finally in section 3.4, we explain the implementation of the OLAP visualization engine.

### **3.1 Mobile Clients in the Client-Centric Architecture**

Mobile clients in the client-centric architecture are divided into two types: app-based and web-based. The app-based mobile client is implemented in Java and is installed in the Android operating system. Android is a popular mobile platform based on the Linux kernel. It includes the Dalvik virtual machine, libraries, middleware, and API's for mobile software development using Java. Figure 13 shows the architecture overview of Android.

An open-source library called kSOAP [22] allows the app-based client to connect to the SOAP web service. After using the web service to download a sub-cube with the lowest level of aggregation, the mobile client is responsible for all computations required by the user, in contrast to the thin client in a server-centric architecture. A screenshot of the app-based client is shown in figure 14.

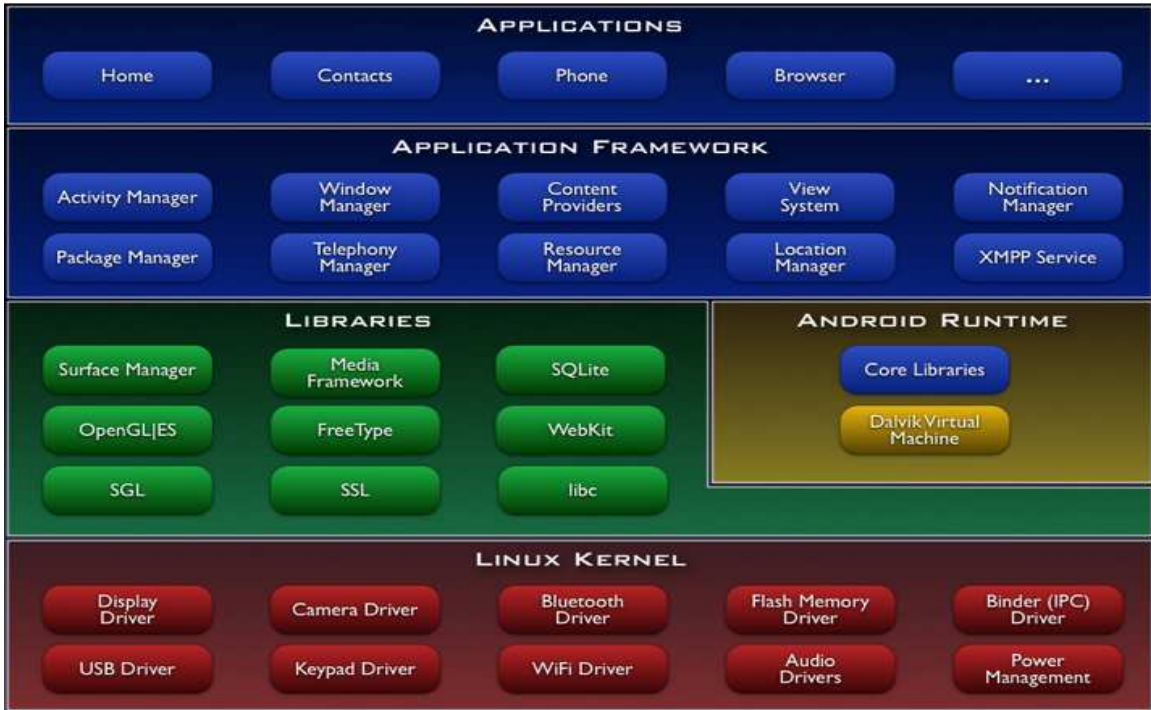


Figure 13. Android architecture Overview [28]

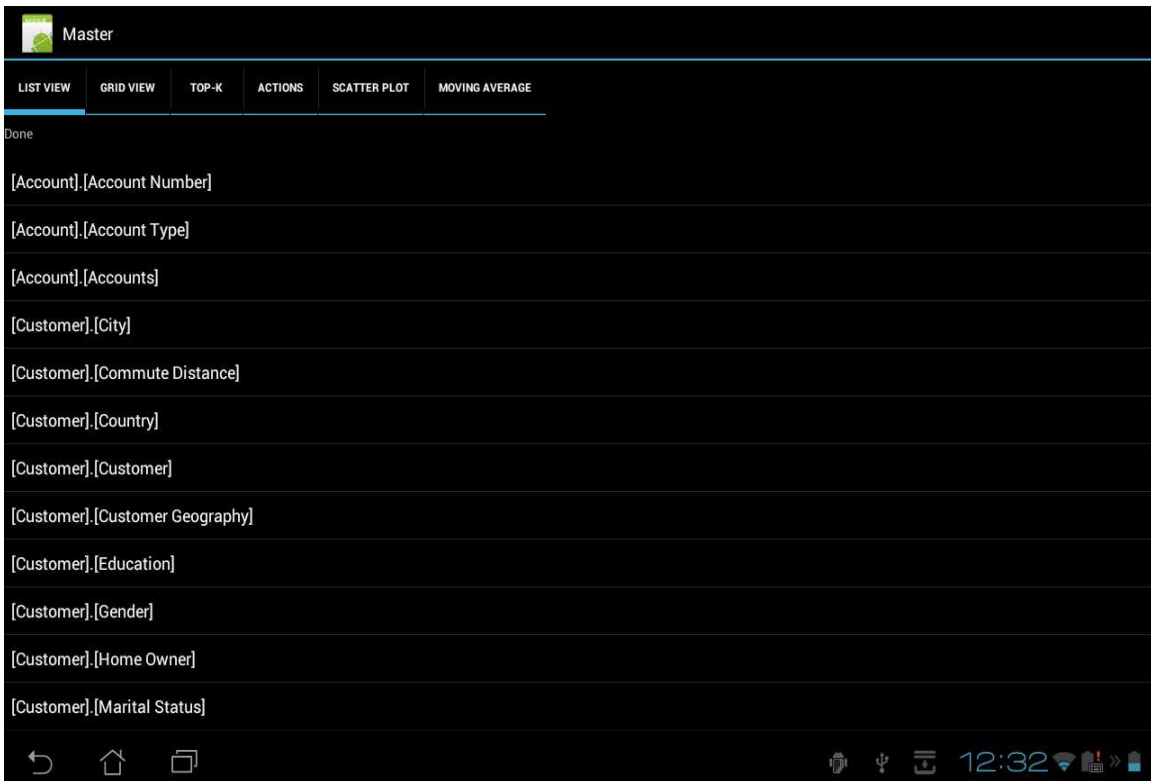


Figure 14. App-based client screenshot



The web-based client is implemented in JavaScript and HTML to run in a web browser and is modified from [27]. For the performance evaluation in chapter 4, we use the Google Chrome web browser for Android. In contrast to the app-based client, the web-based client is not installed in the operating system. The web-based client uses the standard XMLHttpRequest to connect to the SOAP Web Service in order to download the metadata and the cell table. Figure 15 shows a screenshot of the web-based client running in the Chrome browser.

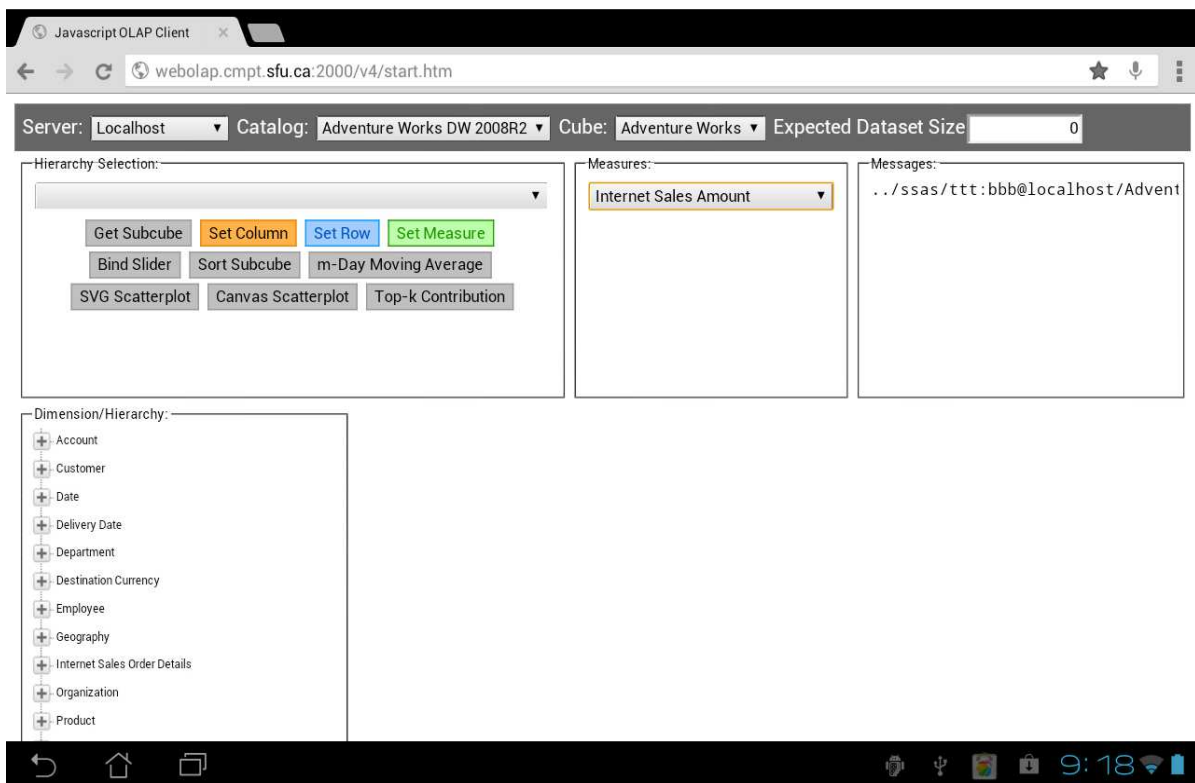


Figure 15. Web-based client screenshot

Since mobile devices such as smartphones and tablets have less memory than powerful desktop machines, it is important for applications to consume less memory. For this reason, Android's Dalvik virtual machine sets an upper limit for heap memory per application. For example, most tablets have at most 48mb of heap memory available for

each application. During runtime, the heap grows dynamically as the Dalvik virtual machine requests more memory when needed.

In contrast to the app-based client, the web-based client running inside a browser is not limited by the Dalvik virtual machine. Therefore, the web-based client has access to more heap memory compared to the app-based client.

The human user begins to use the mobile client by selecting two dimensions from a list. Then the mobile client downloads the metadata and the sub-cube from the web service. Lastly, the mobile client performs OLAP operations using its data engine before presenting the resulting data visualizations to the user.

### **3.2 Mobile Thin Client in the Server-Centric Architecture**

The mobile thin client is implemented in Java as an app just like the app-based client in the client-centric architecture. It also communicates with the server using the same SOAP library. The thin client has the same functions available to the user as the mobile client in the client-centric architecture. However in contrast to the client-centric architecture, the thin client does not download any metadata or sub-cubes. It also does not have a data engine; it only has a visualization engine. Instead, the thin client relies on the data engine of the application server for all computation. Its only job is to present the results from the application server to the human user using its own visualization engine.

### 3.3 OLAP Data Engine

The OLAP data engine implements several common OLAP operations: Cross-tab, drill-down, roll-up, and pivot. We only explain the cross-tab in this section since the other operations are either straightforward or similar to cross-tab.

#### 3.3.1 Cross Tabulation

Cross tabulation (or cross-tab) is an aggregation of a measure based on dimensions. The values from one dimension form the row headers. The values from the second dimension form the column headers. The web service component returns a sub-cube with the lowest level of aggregation. Figure 16 shows this sub-cube represented as a fact table for clarity. An example of a cross-tab on this fact table is displayed in figure 17.

|  | [Geography].[Geog | [Geography].[Geog | [Geography].[Geog | [Geography].[Geog Code].[MEMBER_C | [Customer].[Educat  | [Measures].[Interne Sales Amount] |
|--|-------------------|-------------------|-------------------|-----------------------------------|---------------------|-----------------------------------|
|  | Australia         | New South Wales   | Alexandria        | 2015                              | Bachelors           | 9900142.7571                      |
|  | Australia         | New South Wales   | Alexandria        | 2015                              | Graduate Degree     | 5460560.2513                      |
|  | Australia         | New South Wales   | Alexandria        | 2015                              | High School         | 4638026.0686                      |
|  | Australia         | New South Wales   | Alexandria        | 2015                              | Partial College     | 7723542.8848                      |
|  | Australia         | New South Wales   | Alexandria        | 2015                              | Partial High School | 1636405.2589                      |
|  | Australia         | New South Wales   | Coffs Harbour     | 2450                              | Bachelors           | 9900142.7571                      |
|  | Australia         | New South Wales   | Coffs Harbour     | 2450                              | Graduate Degree     | 5460560.2513                      |

Figure 16. Fact table with the lowest level of aggregation

|                | Bachelors         | Graduate Degree   | High School     | Partial College   | Partial High School |
|----------------|-------------------|-------------------|-----------------|-------------------|---------------------|
| ▶ Australia    | 396005710.284     | 218422410.052     | 185521042.744   | 308941715.392     | 65456210.356        |
| Canada         | 722710421.2682... | 398620898.3449    | 338575903.0078  | 563818630.5904    | 119457583.8997      |
| France         | 475206852.3408    | 262106892.0624    | 222625251.2928  | 370730058.4704    | 78547452.4272       |
| Germany        | 643509279.2114... | 354936416.3345    | 301471694.459   | 502030287.512     | 106366341.8285      |
| United Kingdom | 524707566.1262... | 289409693.3189    | 245815381.6358  | 409347772.8944    | 86729478.72170...   |
| United States  | 3722453676.669... | 2053170654.488... | 1743897801.7936 | 2904052124.684... | 615288377.3463...   |

Figure 17. Cross-tab of Country and Education

The columns are from the Education dimension. The rows are from the Country level of the Geography dimension. The rows in figure 16 are aggregated based on their corresponding values for Country and Education. Rows with identical value for Country and identical value for Education are aggregated. For example, rows where the Country is Australia and the Education is Bachelors are added together to form the same entry in the resulting cross-tab. The resulting cross-tab is an M by N matrix, where M is the number of values in the first dimension and N is the number of values in the second dimension.

Figure 18 shows the algorithm for cross tabulation.

```

Input: levelRow, levelColumn
Output: result table

index=0
rowLength=rows.length
colLength=columns.length

For i=0 to rowLength-1
  {
    currentRow=rows[i]
    if(levelRow>currentRow[0])
    {
      index+=colLength
      continue
    }
    row=currentRow[levelRow+1]*colLength

    For j=0 to colLength-1
    {
      currentColumn=columns[j]
      if(levelColumn<currentColumn[0]) //index 0 contains the number of indexes
      {
        resultTable[row + currentColumn[levelColumn+1]]+=CellTable[index]
      }
      index++
    }
  }
}

```

Figure 18. Cross-tab algorithm

The inputs are the levels of the cross-tab. For example in the cross-tab shown by figure 17, Country is level 1 in the Geography dimension's hierarchy and Education is level 1 in the Education hierarchy. Figure 19 shows a cross-tab of State-Province and Education. State-Province is level 2 in Geography's hierarchy and Education is still level 1.

|                   | Bachelors      | Graduate Degree | High School    | Partial College | Partial High School |
|-------------------|----------------|-----------------|----------------|-----------------|---------------------|
| ▶ New South Wales | 188102712.3849 | 103750644.7747  | 88122495.3034  | 146747314.8112  | 31091699.9191       |
| Queensland        | 79201142.0568  | 43684482.0104   | 37104208.5488  | 61788343.0784   | 13091242.0712       |
| South Australia   | 29700428.2713  | 16381680.7539   | 13914078.2058  | 23170628.6544   | 4909215.7767        |
| Tasmania          | 9900142.7571   | 5460560.2513    | 4638026.0686   | 7723542.8848    | 1636405.2589        |
| Victoria          | 89101284.8139  | 49145042.2617   | 41742234.6174  | 69511885.9632   | 14727647.3301       |
| Alberta           | 19800285.5142  | 10921120.5026   | 9276052.1372   | 15447085.7696   | 3272810.5178        |
| British Columbia  | 287104139.9559 | 158356247.2877  | 134502755.9894 | 223982743.6592  | 47455752.5081       |

Figure 19. Cross-tab of State-Province and Education

This algorithm uses indexes, downloaded from the web service component, to obtain a quick reference to the result table. Each leaf of the hierarchy tree contains the references. Figure 20 shows the indexes for the Education dimension. When the level is 1 for Education, there are five columns in the result table. The indexes are 0,1,2,3, and 4. When the level is 0 for Education, there is 1 column in the result table. As a result, the index is 0 for all leaves.

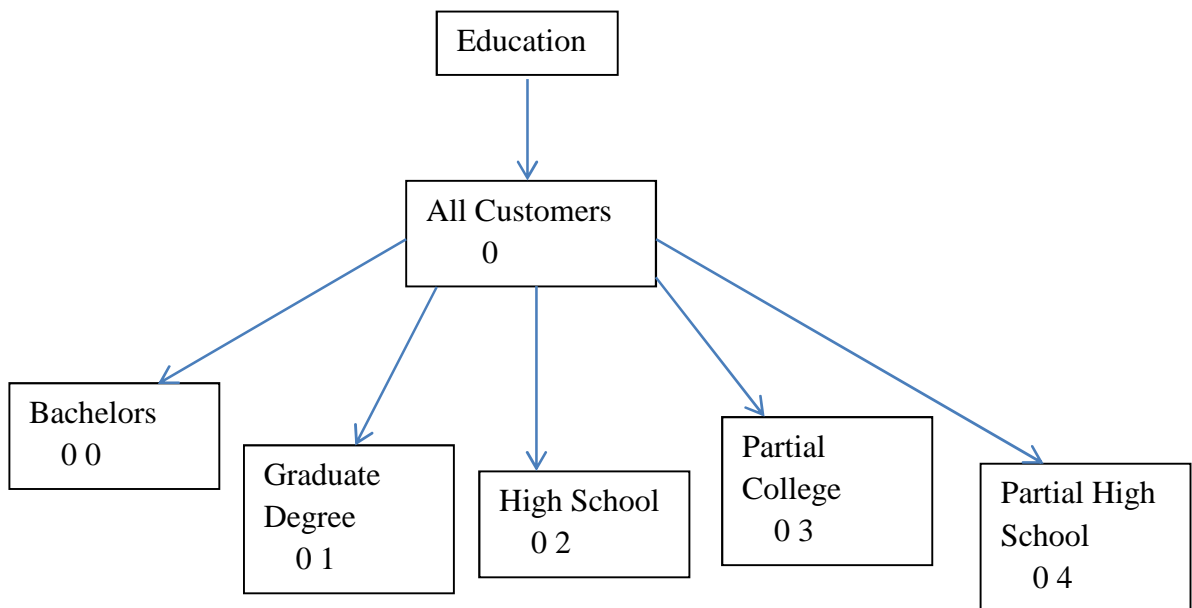


Figure 20. Metadata indexes.

### 3.3.2 Parallel Cross Tabulation

The newest generation of smartphones and tablets has dual-core or quad-core processors. The ASUS Transformer Prime has a quad-core processor capable of running multi-threaded applications in parallel. Computing the cross tabulation in parallel may lead to further improvements in performance for the app-based client.

Fortuna et al. conducted a study on the potential parallelism of JavaScript applications by examining dependency types and looping behavior [29]. The potential speedup averages 8.9 times and as high as 45.5 times. Unfortunately, there is no working implementation of data parallelism for JavaScript. HTML5 Web Workers [30] are independent processes that uses message passing for communication, instead of shared memory. Because the messages passed are copied, not shared, HTML5 Web Workers are unsuitable for parallelizing the cross-tab function. In addition, Intel Lab's River Trail project enables data parallelism in JavaScript web applications [31]. However, the project is currently at the prototype stage, and is not compatible with Android. Furthermore, WebCL is a JavaScript binding for heterogeneous parallel computing [32]. It is currently in development, and thus is unavailable for Android. For the above reasons, the web-based client is currently unable to fully utilize the hardware of mobile devices, in contrast to the app-based client.

Parallelization of cross tabulation for the app-based client can be done using the standard Thread library from Java. The algorithm for parallel cross-tab is shown in figure 21. The cell table is divided by the number of threads. Then each thread computes the cross-tab on a segment of the cell table. The parallel version of the cross-tab algorithm

consumes more memory than the serial version since each thread must have its own result table.

```
Input: levelRow, levelColumn, num
Output: result table

rowLength=rows.length
colLength=columns.length
size=rowLength/threads;
if(num==threads-1)
{
    start=num*size
    end=rowLength
}
else
{
    start=num*size
    end=start+size
}
index=start*colLength

For i=start to end
{
    currentRow=rows[i]
    if(levelRow>currentRow[0])
    {
        index+=colLength
        continue
    }
    row=currentRow[levelRow+1]*colLength
    For j=0 to colLength-1
    {
        currentColumn=columns[j]
        if(levelColumn<currentColumn[0])
        {
            resultTable[row +
                currentColumn[levelColumn+1]]+=CellTable[index]
        }
        index++
    }
}
}
```

Figure 21. Parallel cross-tab algorithm



## 3.4 OLAP Visualization Engine

This section explains how the results of the data engine are presented to the user. After the computation of the cross-tab is completed, the results are displayed in a data grid. Section 3.4.1 explains the implementation of the data grid. However, advanced OLAP users require data visualizations that are more complex than a simple data grid. The sections from 3.4.2 to 3.4.4 explain three graphical data visualizations. These three graphical data visualizations are top-k percentage contribution, moving average, and scatter plot aggregation. Graphical data visualizations for the app-based client are implemented using AChartEngine [23], a Canvas graphical library for Android. Graphical data visualizations for the web-based client are implemented using vector-graphic based libraries and the new HTML5 Canvas library.

### 3.4.1 Data Grid Implementation

This section describes the implementation of the data grid. Figure 22 shows a sample data grid created by the app-based client.

If the data is massive, slow rendering will block the user interface until it is completed. For this reason, the data grid is implemented using on-demand paging for both the web-based client and the app-based client. When the data grid is large, not all rows or columns are visible at the same time. When the user scrolls the data grid on a mobile device, new rows become visible, and certain rows disappear. Because creating a new row is an expensive operation, the data grid recycles the old rows that are no longer visible. The old rows are loaded with new data, and thus the application saves memory and CPU consumption by using a small data grid to display the large amounts of data.

The web-based client implements the data grid using HTML table, and the app-based client implements the data grid using TextViews [33].

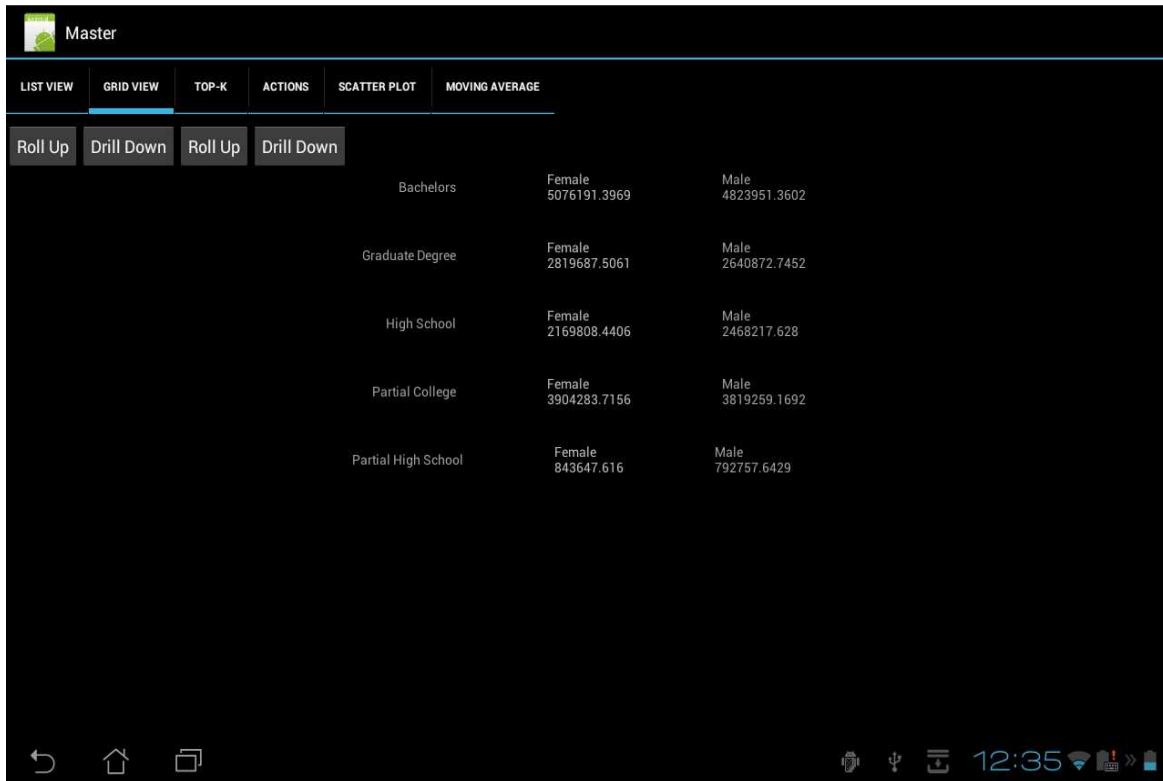


Figure 22. Data grid

### 3.4.2 Top-K Percentage Contribution

Top-k percentage contribution ranks each value's contribution to the total sum of all the values during a certain time period. Then it returns the k values with the highest contributions. A realistic scenario of using top-k is the following. A user wants to find the top 10 best-selling products and their percentage contribution to the sales total during the last 30 days. First, the sales amount of each product during the last 30 days is aggregated. Then the running total of the sales amount of each product is ranked and the top 10 best-selling products are returned to the user.

The result of the top-k computation is displayed in a table, as well as a pie chart. The web-based client renders the pie chart using *Google Chart Tools*, a SVG library [34]. The app-based client uses aChartEngine [23]. A slider allows the user to change the time period. Every movement of the slider triggers a function to re-computes the top-k percentage contribution. This function first computes a cross-tab on the sub-cube given the selected time period, and then ranks each value's contribution to the total sum of all the values. Because each movement of the slider triggers this function, the user may generate a large number of calculations in a short time period. Figure 23 shows what the results look like for top-k percentage contribution on the app-based client.

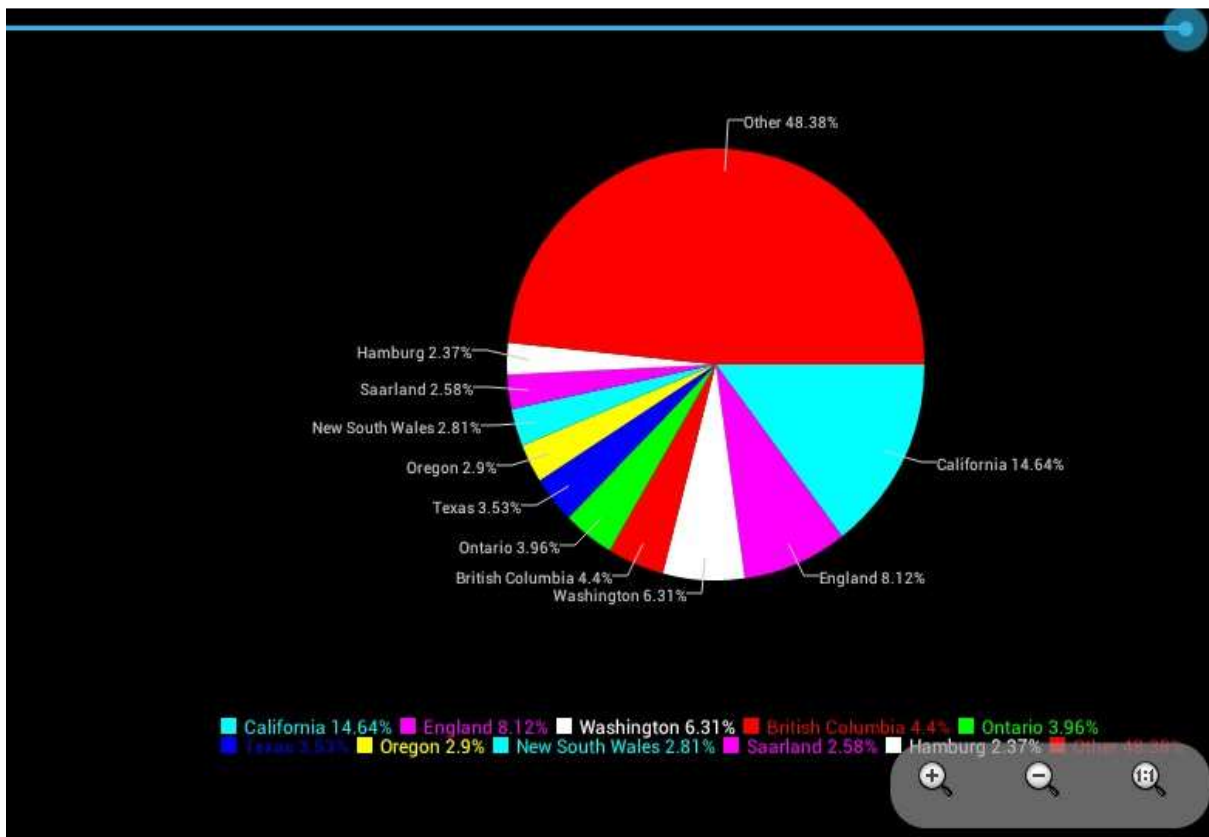


Figure 23. Top-k screen shot

### 3.4.3 Moving Average

The second graphical data visualization is moving average. A moving average analyzes a set of data by computing a series of averages of subsets of the data. Moving average is used in analysis of economic data. It is also used in financial applications.

The moving average we use is the simple moving average, which is the un-weighted average of the previous  $n$  data points. We are given a series of numbers and a subset with a fixed size. The first moving average is computed as the average of the first subset of the number series. Then the subset is shifted forward to include a new data point and to exclude the first data point in the series. A new average is computed. This process repeats until the average of the last subset of the data series is computed.

For example, we want the simple un-weighted moving average in an  $n$ -day subset of sales amount. The first element in the moving average is the average sales amount of the first  $n$  days. If are the sales amounts for the first  $n$  days, then the formula for the first moving average is

$$SMA_{first\ day} = \frac{\sum_{i=1}^n S_i}{n}$$

Calculating the successive values is an inexpensive and simple operation. A new value is added to the previous summation and the oldest value is removed to form a new summation.

$$SMA_{today} = SMA_{last\ day} - \frac{S_{today-n}}{n} + \frac{S_{today}}{n}$$

Moving average is implemented as a line graph using *flot*, a Canvas library for the web-based client [35]. The app-based client uses aChartEngine [23]. An example of a simple moving average created by the web-based client is shown in figure 24.

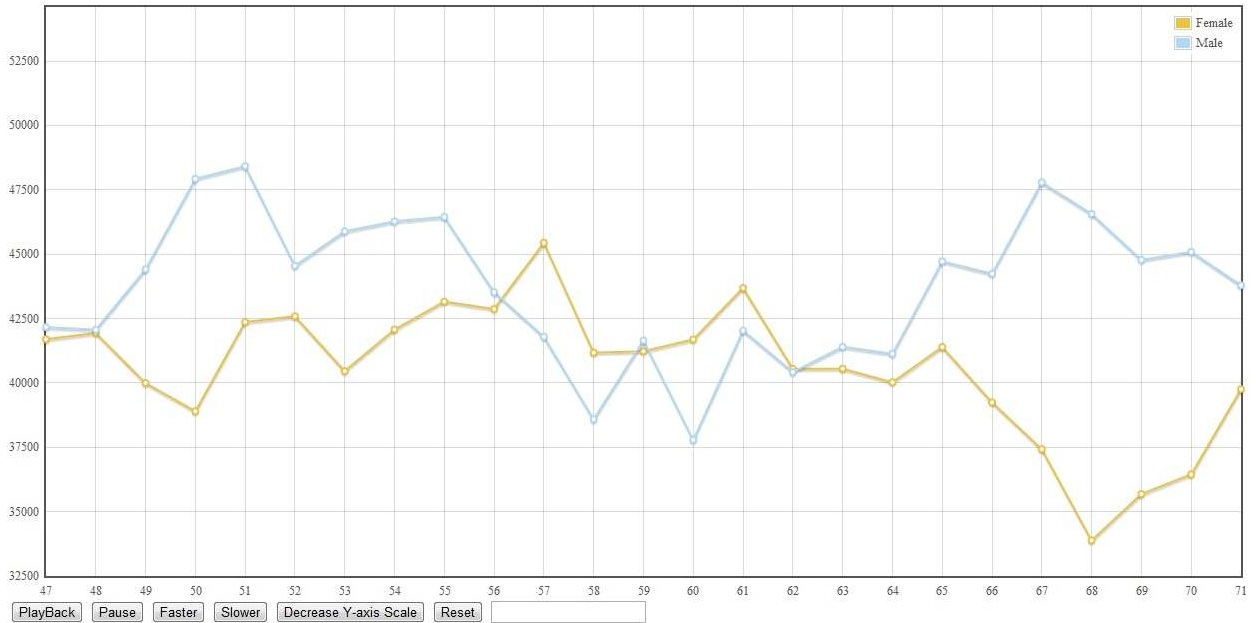


Figure 24. Moving average screen shot

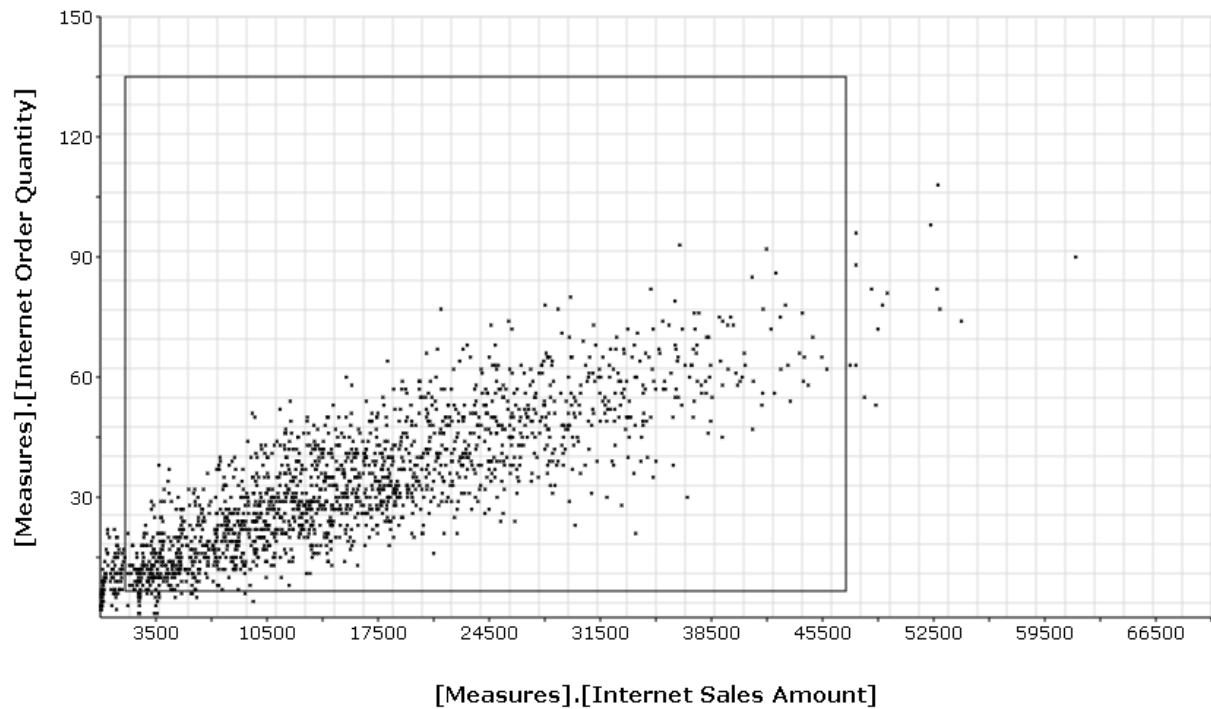
A playback function computes the moving average for the next day, and presents an animation as a way to observe trends in the moving averages of the data series. When the user starts the animation, the OLAP client calls the playback function to compute the moving average for the next day and continues to redraw the animation until it reaches the end of the data series.

### 3.4.4 Scatter Plot

Although data grid rendering performance is not a major problem when on-demand paging is implemented as shown by a previous section, the author in [24] argues that the amount of data in a large data grid can cause OLAP analysts to ignore important and interesting trends in the data. Graphical visualization tools provide an alternative platform to present OLAP data cubes and data [25, 26].

Scatter plot is a simple two variable graphical visualization tool for a set of data. The data is displayed as a series of points such that the position on the scatter plot is

determined by the values for the two variables. The value of one variable determines a point's position on the horizontal axis and the value of the second variable determines a point's position on the vertical axis. The scatter plot is very useful for displaying the correlation of two variables. The correlations may be positive, negative, or the correlation does not exist. For OLAP, the two variables are two measures. An example of the scatter plot created by the web-based client is displayed in figure 25. The web-based client renders scatter plots by using *Protovis*, a SVG library [7]. The app-based client uses *aChartEngine* [23].



|  |             |
|--|-------------|
| Number of points:  | 1824        |
| Number of selected points:                               | 1691        |
| Aggregation on X ([Measures].[Internet Sales Amount]):   | 28382528.25 |
| Aggregation on Y ([Measures].[Internet Order Quantity]): | 58104.00    |
| Aggregation on Z (undefined):                            | NaN         |
| Time (in ms):  | 1           |

Figure 25. Scatter plot screen shot

An aggregation function has been implemented for the scatter plot by both the web-based and app-based clients. For example, the user drill-down on a region in the scatter plot by touch of the finger. When the region changes in size and position, the function aggregates the data points that are positioned within the user selected region on their x and y coordinates. This type of data analysis provides the ability to rapidly determine the contribution of a region of visible outliers in the data, and compare it with other selected regions in the scatter plot.

## **4. PERFORMANCE EVALUATION**

In this chapter, we perform a performance evaluation of the client-centric architecture in section 4.1 and a performance evaluation of the server-centric architecture in section 4.2. The thin client in the server-centric architecture is app-based and we compare it with the app-based client of the client-centric architecture. The web-based client runs in the Google Chrome for Android web browser. The measure is [Internet Sales Amount].

The mobile device used for the performance evaluation is an ASUS Transformer Prime [8] tablet. This tablet has a 1.3 GHZ quad-core Nvidia Tegra 3 processor, and 1GB of main memory. Each core has 32KB L1 cache. The four cores share a 1MB L2 cache. It runs version 4.0.3 of the Android operating system. The server is equipped with four quad-core Intel Xeon E5420 processors and 4GB of memory. The web service is ASP .Net. The OLAP server is Microsoft Analysis Services 2008 R2.

### **4.1 Client-Centric Performance**

In this section, we compare the app-based client and the web-based client on metadata download and parsing times, sub-cube download and parsing times, computing a cross-tab (a common OLAP operation), parallelization, data grid rendering, and graphical data visualizations.

#### **4.1.1 Metadata Download and Parsing**

After the human user selects a dimension, the mobile client sends a request for the metadata to the web service. After a dimension's metadata is downloaded, it is parsed. In



this section, we measure the download times of the metadata, and evaluate the parsing times of the app-based and web-based clients. We use ten dimensions from the *Adventure Works* data cube. The dimensions are [Geography].[Geography], [Employee].[Employee], [Date].[DayOfYear], [Date].[Calendar],[Reseller].[ResellerBank], [Customer].[Education], [Reseller].[BankName], [Reseller].[OrderMonth], [Empolyee].[DepartmentName], and [Empolyee].[Employee Department], respectively.

|               | Download Time (ms) | Web-Based Parse Time (ms) | App-Based Parse Time (ms) |
|---------------|--------------------|---------------------------|---------------------------|
| Dimension #1  | 994                | 14                        | 9                         |
| Dimension #2  | 915                | 6                         | 4                         |
| Dimension #3  | 866                | 7                         | 5                         |
| Dimension #4  | 967                | 104                       | 33                        |
| Dimension #5  | 922                | 59                        | 6                         |
| Dimension #6  | 584                | 3                         | 0                         |
| Dimension #7  | 609                | 1                         | 0                         |
| Dimension #8  | 526                | 0                         | 0                         |
| Dimension #9  | 636                | 0                         | 1                         |
| Dimension #10 | 871                | 13                        | 4                         |

Table 2. Meta information

Table 2 shows the download time and parsing time in milliseconds for each dimension. The download time is the same for both clients since they share the same web service and OLAP server. The app-based is able to parse the metadata faster than the

web-based client. However, because the download time is much greater in magnitude than the parsing time, the parsing advantage of the app-based client is not significant overall.

The download can be reduced in two ways. First of all, the SOAP Web Service could store the most recently requested metadata in a cache. Secondly, the metadata could be stored on the mobile client's internal storage.

#### **4.1.2 Sub-Cube Download and Parsing**

After the metadata are downloaded from the web service, the mobile client must download a sub-cube as a cell table from the web service. After a sub-cube is downloaded, it is parsed from a string into an array of floating point values. In this section, we measure the download times of the sub-cubes, and evaluate the parsing times of the app-based and web-based clients. We use five sub-cubes from the *Adventure Works* data cube.

The first sub-cube is composed of the dimensions [Date].[Day of Year] and [Geography].[Geography], which has 90885 tuples. The second sub-cube is composed of the dimensions [Geography].[Geography] and [Employee].[Employees], which has 163095 tuples. The third sub-cube is composed of the dimensions [Reseller].[ResellerBank] and [Employee].[Employees], which has 174549 tuples. The fourth sub-cube is composed of the dimensions [Geography].[Geography] and [Employee].[Employee Department], which has 193880 tuples. Finally, the fifth sub-cube is composed of the dimensions [Date].[Calendar] and [Employee].[Employees], which has 295812 tuples.

Table 3 shows the download time and parsing time in milliseconds for each sub-cube. The download time is the same for both clients since they share the same web service and OLAP server. The app-based is able to parse the cell table about 2 to 3 times faster than the web-based client. However, because the download time is much greater in magnitude than the parsing time, the parsing advantage of the app-based client is not significant overall.

The download can be reduced in several ways. First of all, a number of OLAP servers have the option of optimizing at leaf level pre-aggregation. Second, the SOAP Web Service could store the most recently requested sub-cubes in a cache. Finally, the sub-cube could be stored on the mobile client's internal storage.

|             | Tuples | Download Time (ms) | Web-Based Parse Time (ms) | App-Based Parse Time (ms) |
|-------------|--------|--------------------|---------------------------|---------------------------|
| Sub-Cube #1 | 90885  | 6315               | 214                       | 93                        |
| Sub-Cube #2 | 163095 | 8772               | 489                       | 380                       |
| Sub-Cube #3 | 174549 | 11397              | 483                       | 182                       |
| Sub-Cube #4 | 193880 | 10803              | 537                       | 203                       |
| Sub-Cube #5 | 295812 | 15274              | 831                       | 215                       |

Table 3. Sub-cubes information

#### 4.1.3 Cross Tabulation Evaluation

In this section, the web-based client is compared with the app-based client on the cross-tab computation. Performance is measured by the running time in milliseconds.

Each experiment is run at least five times. Then, the average running time is used for the

comparison. The performance of the two clients is compared on five sub-cubes of different sizes and hierarchical structures from the *Adventure Works* data cube. Detailed information on the five sub-cubes is found in table #3 in section 4.1.2.

The performance results are shown in figure 26. From the results, we can see that the app-based client computes a cross-tab significantly faster than the web-based client. Figure 27 shows the speedup gained by the app-based client. The running time of the app-based client is normalized with respect to the running time of the web-based client. The app-based client is around two times faster than the web-based client for all five sub-cubes. This is due to Java performing calculations faster than JavaScript.

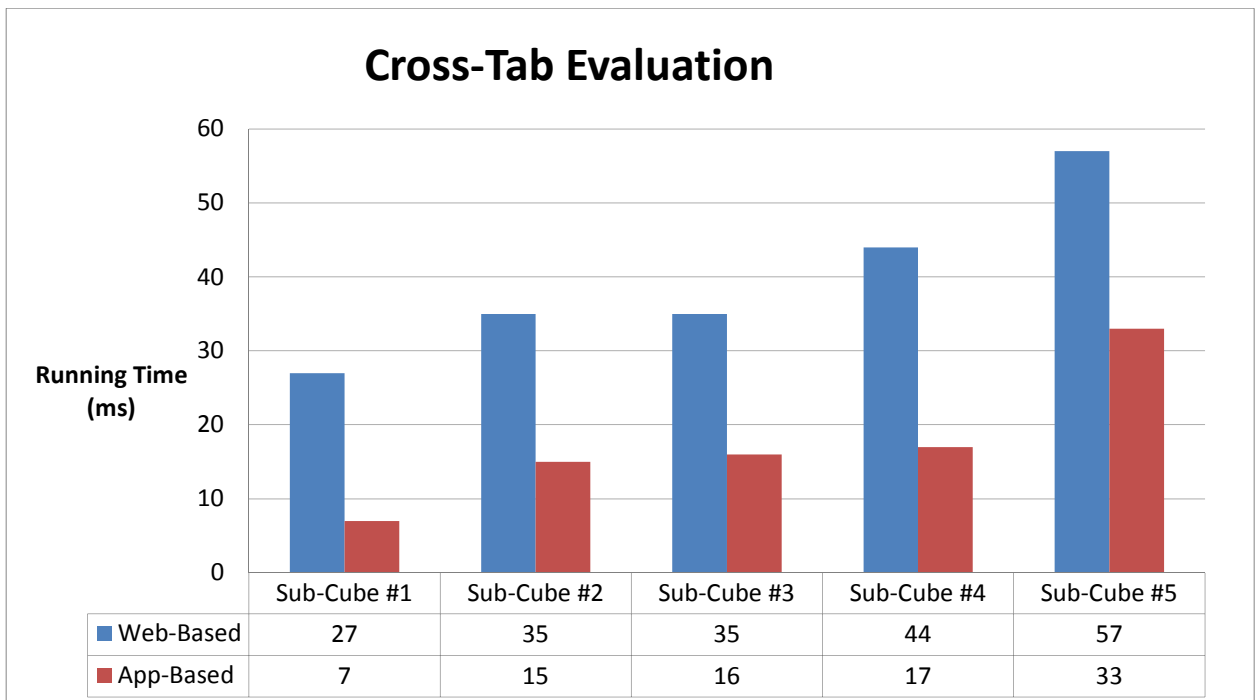


Figure 26. Cross-tab performance comparison

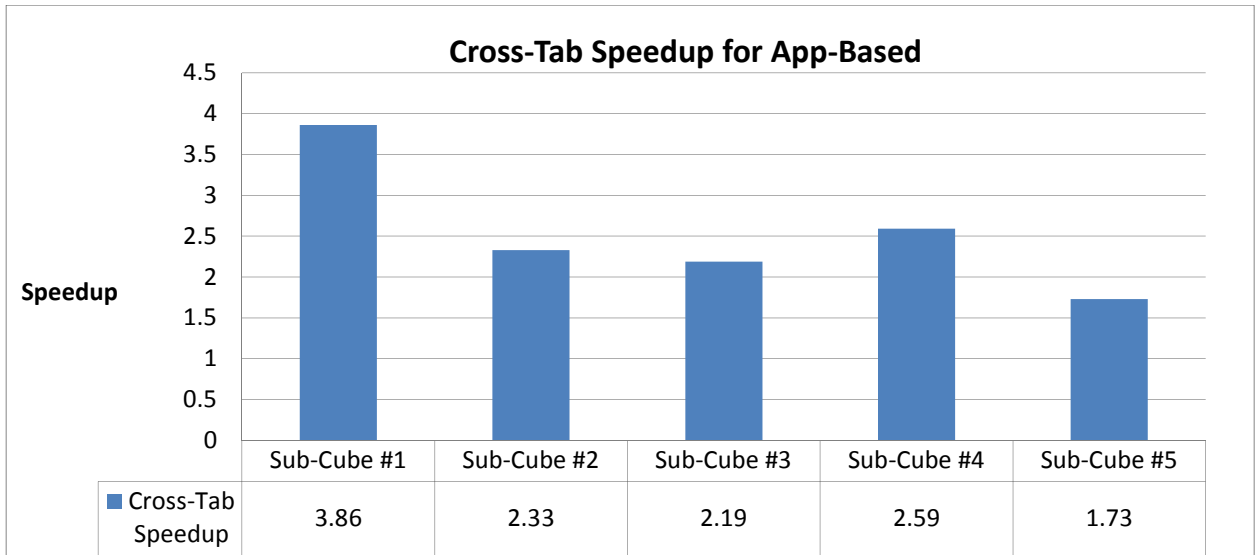


Figure 27. Cross-tab speedup

#### 4.1.4 Parallel Cross-Tab Evaluation

To evaluate the parallel version of cross-tab, we created three large sub-cubes. The first sub-cube has one million tuples. The second sub-cube has two million tuples. And the third sub-cube has three million tuples. Each tuple has the numerical value of 999999.99. We first run the serial algorithm from figure 18 and then the parallel cross-tab algorithm with three different values for the number of threads: two, three, and four. The running times, measured in milliseconds, are shown in figure 28 for the app-based client.

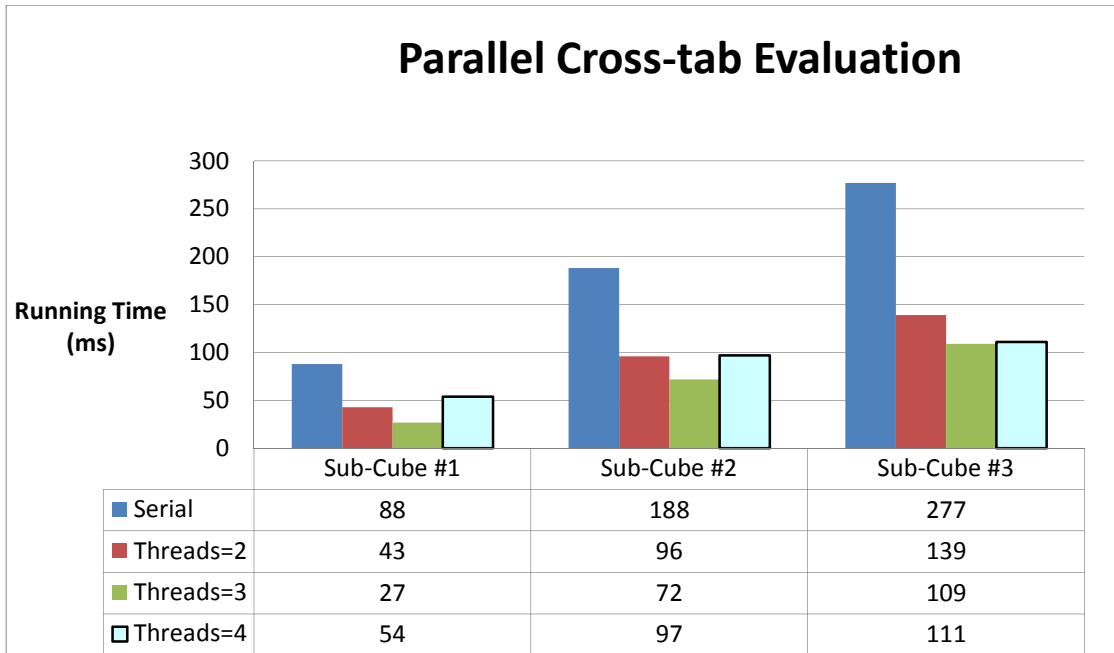


Figure 28. Cross-tab parallel performance

The speedup achieved by parallelization is displayed in figure 29. Computing the cross-tab in parallel can result in performance that is about three times faster than the serial algorithm for all three sub-cubes. For the first sub-cube in particular, the speedup is super linear when the number of threads is two or three.

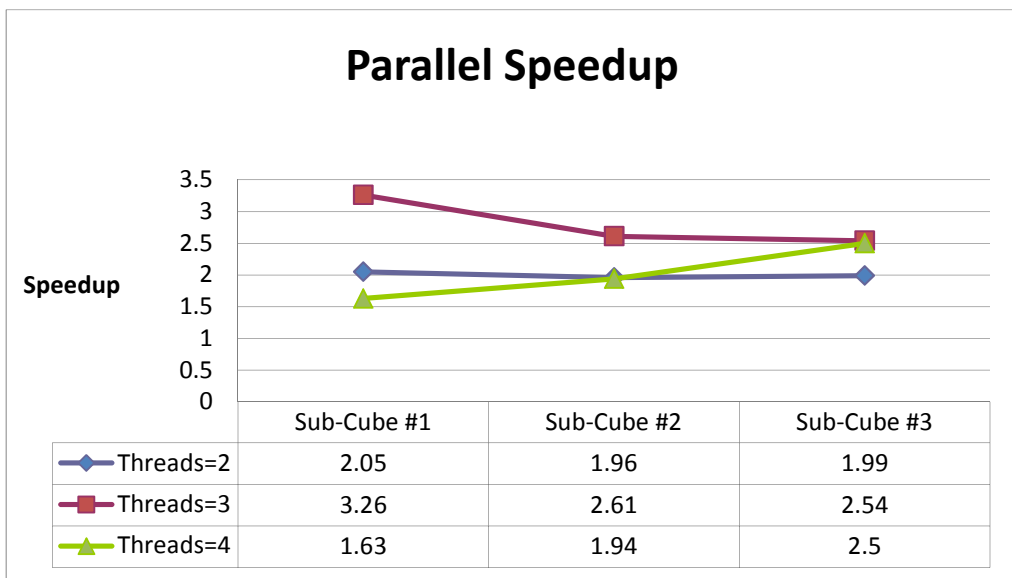


Figure 29. Cross-tab parallel speedup

Figure 30 shows the efficiency of the parallel cross-tab algorithm. Parallel efficiency is defined as the speedup divided by the number of threads. When the number of threads is two, parallel efficiency is high for all three sub-cubes. Increasing the number of threads results in efficiency degradation, which is caused by contention for shared memory resources and communication bottlenecks.

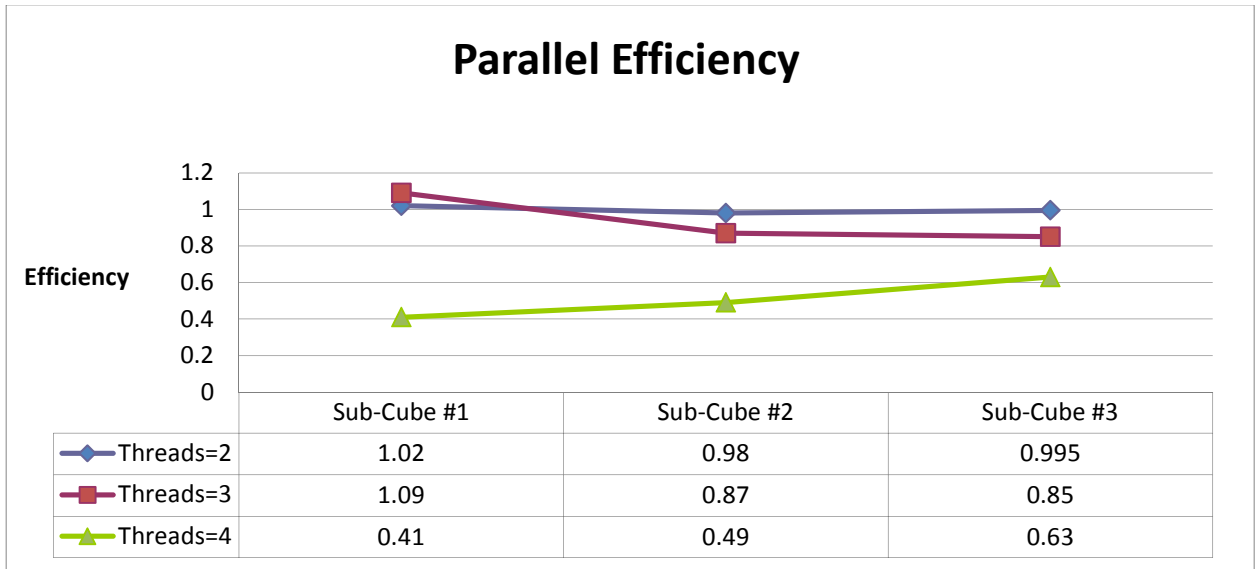


Figure 30. Cross-tab parallel efficiency

Since the web-based client is unsuitable for data parallelism, we use its serial running time to compare it with the best running time of the app-based client. The running times in milliseconds are shown in figure 31 and the overall speedup for the app-based client over the web-based client is shown in figure 32. Overall, the app-based client is up to 4.81 times faster than the web-based client.

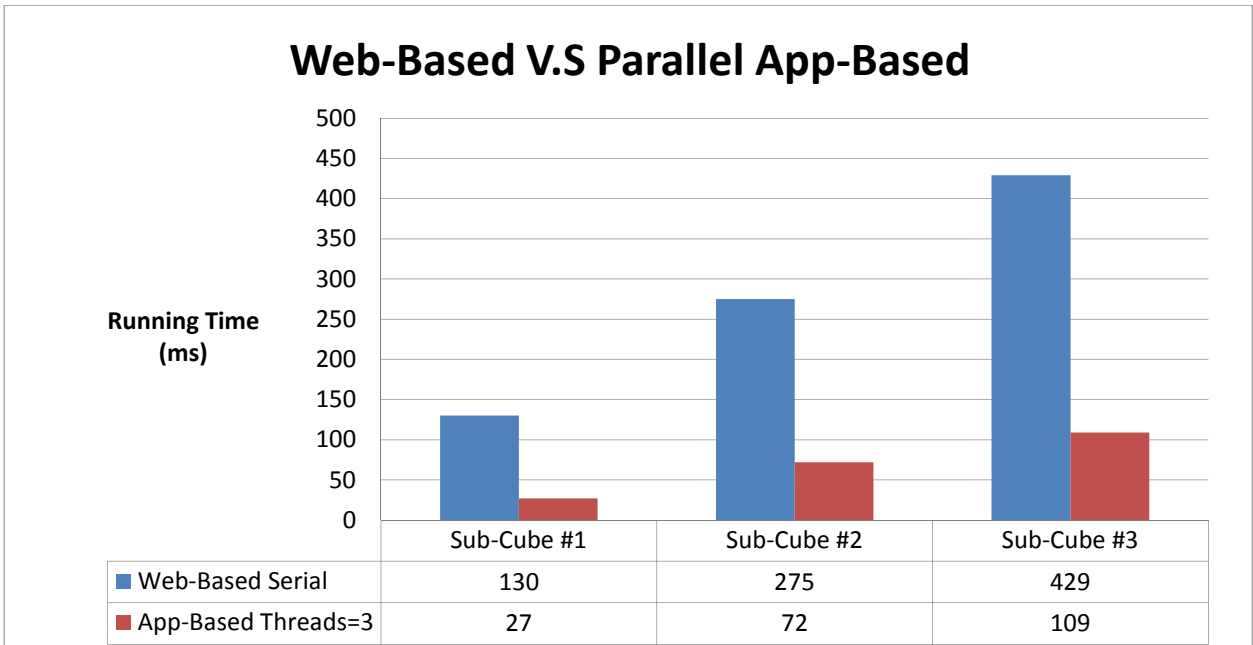


Figure 31. Web-based and parallel app-based cross-tab performance comparison

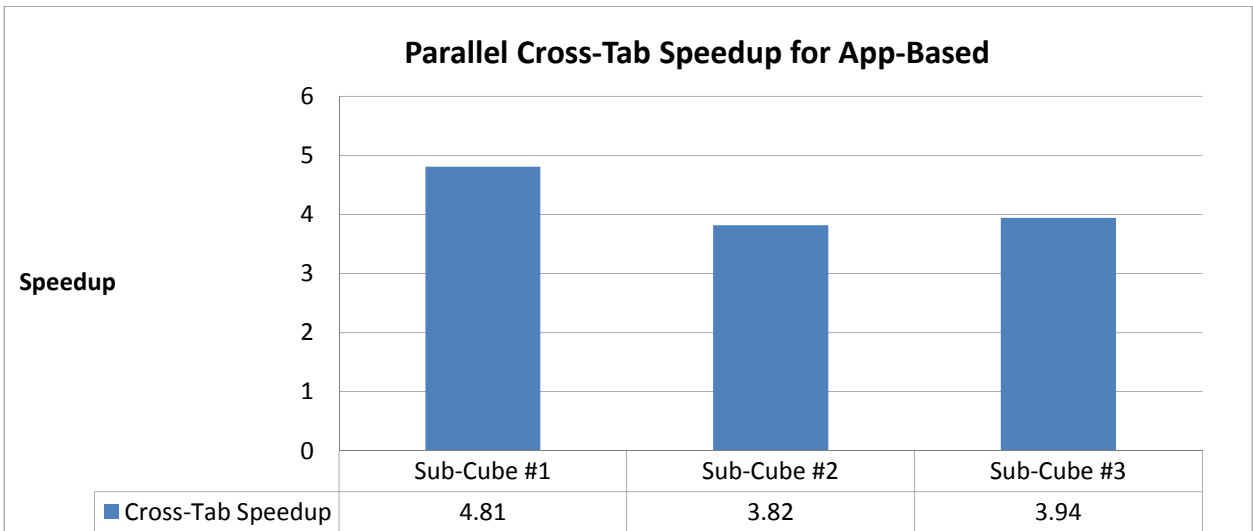


Figure 32. Parallel cross-tab speedup for app-based

#### 4.1.5 Data Grid Evaluation

In this section, we compare the web-based and app-based clients on rendering the results of a cross-tab in a 2-dimensional grid. The web-based client requires 20 milliseconds to create a 6 by 9 grid, and the app-based client requires 30 milliseconds to



create the same grid. After creating the grid, the web-based client requires 3 milliseconds to load the data to the grid, and the app-based client requires 10 to 20 milliseconds to load the data to the grid. The reason loading the data is a slow operation for the app-based client is because setting the text of a TextView is slow. Therefore, the web-based is superior in displaying cross-tab results in a grid compared to the app-based client.

#### **4.1.6 Top-k Percentage Contribution Evaluation**

To evaluate the app-based client against the web-based client, we perform an experiment with three different values for  $k$  and the sub-cube with the dimension [Geography].[Geography] and the time dimension [Date].[DayOfYear] from the *Adventure Works* data cube. This sub-cube has 239075 tuples. The download time is 11466 milliseconds. The web-based client used 647 milliseconds to parse the sub-cube, and the app-based client used 200 milliseconds. The slider is moved from left to right to adjust the time period, and each slider movement recalculates the output. The average running time in milliseconds of the calculations is used for the comparison of the two OLAP clients. The performance results of the evaluations are displayed in figure 33 and figure 34 shows the speedup gained by the app-based client.

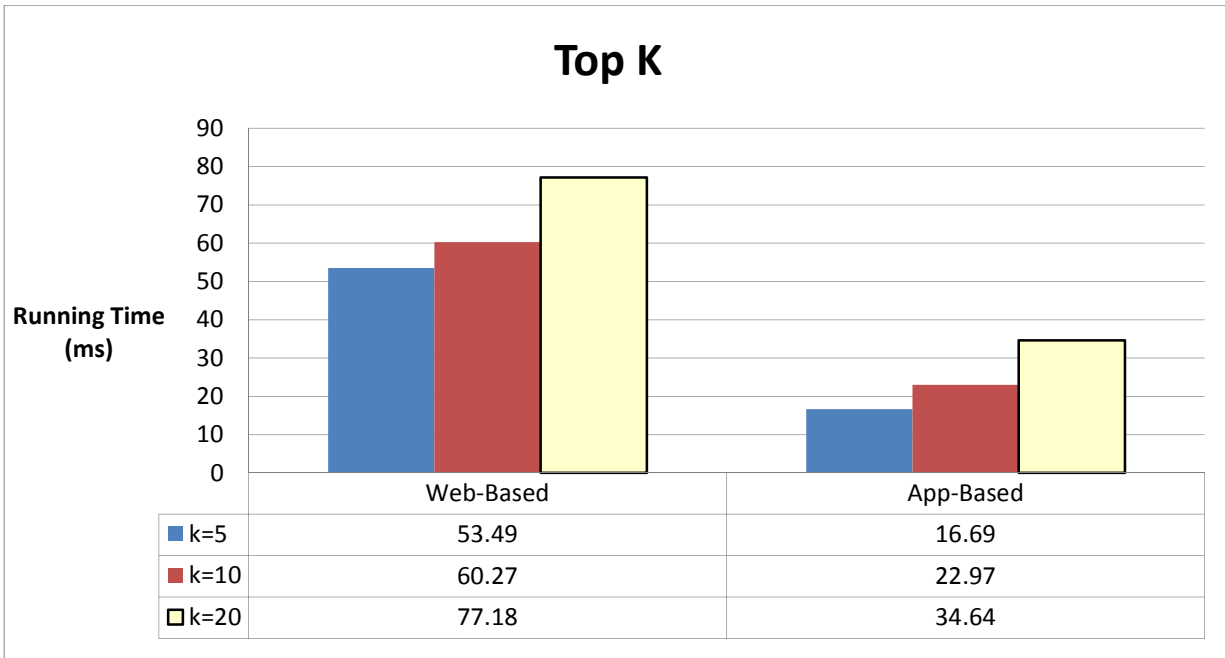


Figure 33. Top-k performance comparison

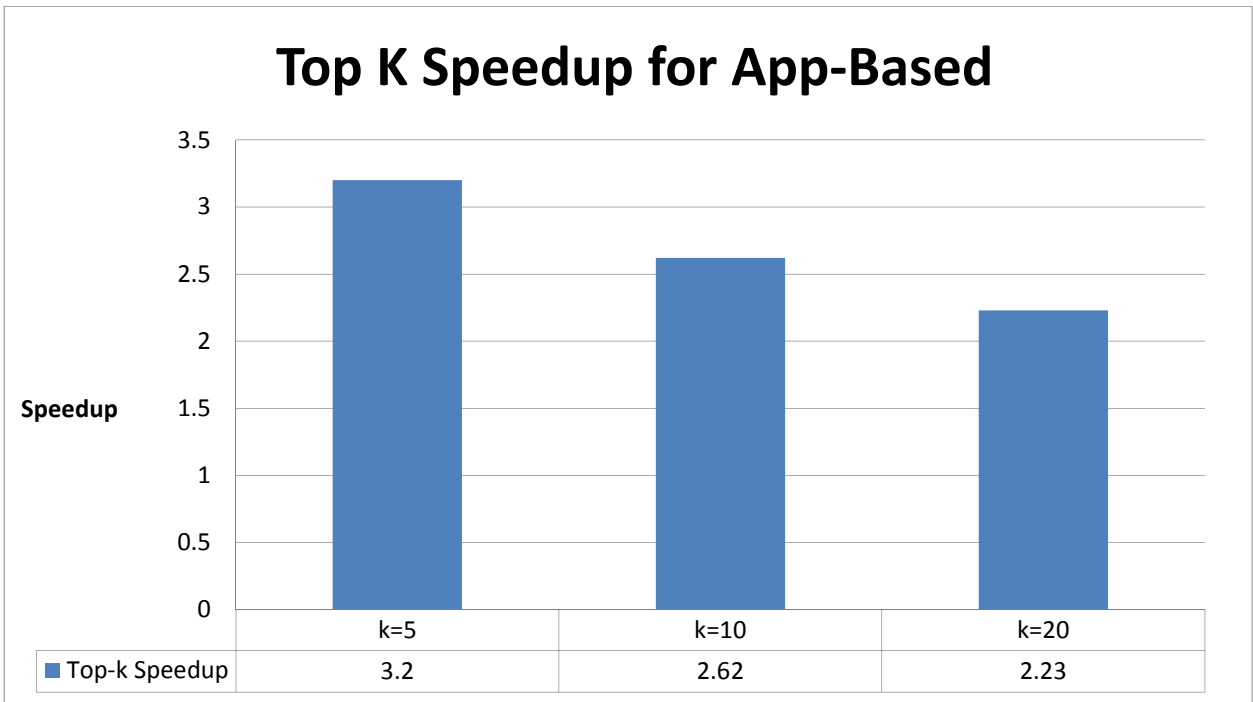


Figure 34. Top-k speedup

From the results, we can see that the app-based client computes and renders the top-k percentage contribution faster than the web-based client. The running time of the app-based client is normalized with respect to the running time of the web-based client in figure 34. The app-based client is two to three times faster than the web-based client for all three values of k.

Table 4 shows the breakdown of the top-k running time in milliseconds. Rendering the pie chart is much slower for the web-based client, resulting in worse visualization smoothness compared to the app-based client. The rendering superiority of the app-based client is due to implementation of the Android graphics.

|                          | Web-Based | App-Based |
|--------------------------|-----------|-----------|
| Pie Chart Rendering (ms) |           |           |
| k=5                      | 36.41     | 7.18      |
| k=10                     | 40.84     | 12.59     |
| k=20                     | 54.29     | 23.38     |
| Top-K Calculation (ms)   |           |           |
| k=5                      | 17.08     | 9.51      |
| k=10                     | 19.43     | 10.38     |
| k=20                     | 22.89     | 11.26     |
| Total Running Time (ms)  |           |           |
| k=5                      | 53.49     | 16.69     |
| k=10                     | 60.27     | 22.97     |
| k=20                     | 77.18     | 34.64     |

Table 4. Top-k running time break down

#### 4.1.7 Moving Average Evaluation

To compare the web-based client and the app-based client on computing moving averages, we use the sub-cube with the dimensions [Customer].[Education] and [Date].[Day of Year] from the *Adventure Works* data cube. This sub-cube has 1825 tuples and requires 511 milliseconds to download. The web-based client requires 4 milliseconds to parse this sub-cube, and the app-based client requires 3 milliseconds. We use three variations for n, the size of the subset of the data series: 10 days, 20 days, and 30 days. The playback function is invoked a number of times to calculate each moving average. The average performance of the playback function, including the rendering time, is used to make a comparison of the two clients. The results, in milliseconds, are shown in figure 35.

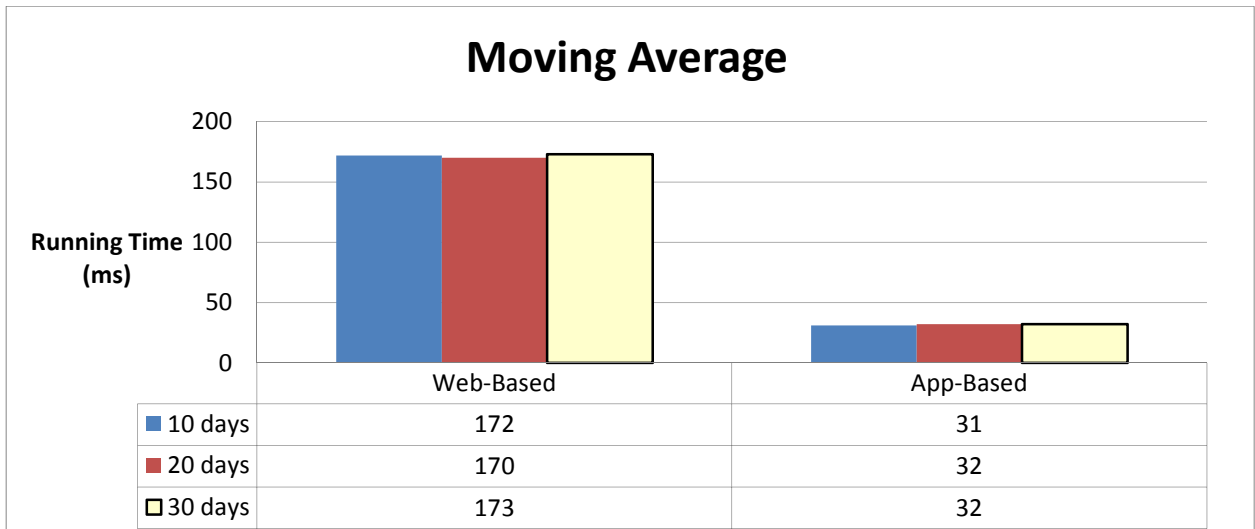


Figure 35. Moving average performance in milliseconds

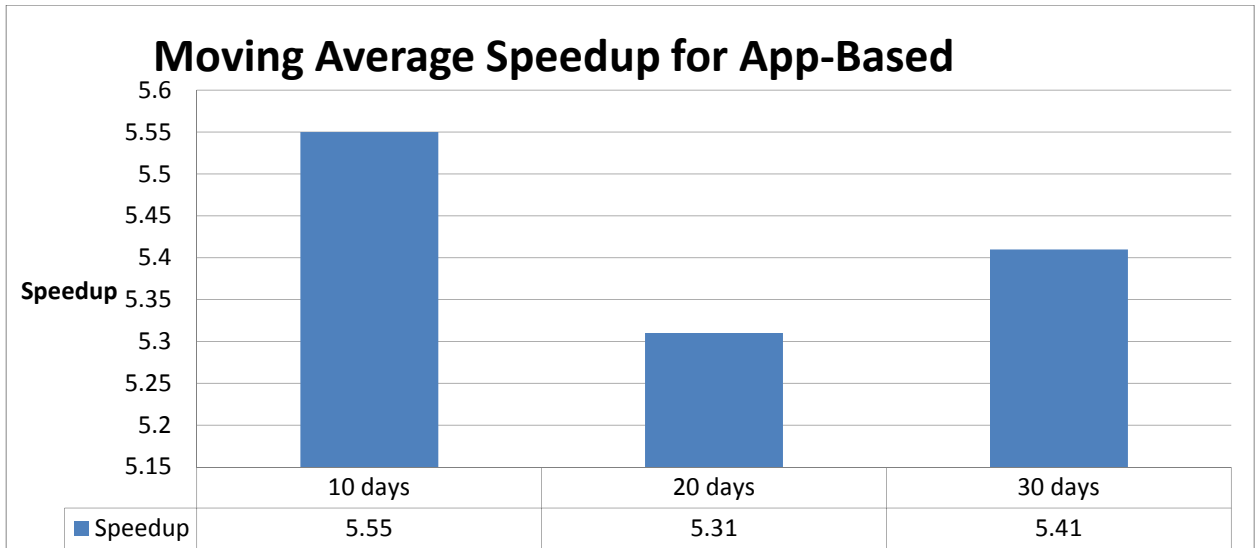


Figure 36. Moving average speedup

It is clear that the app-based client is five times faster than the web-based client for all three subset sizes as shown in figure 36. The app-based client computes a moving average and renders the result in 32 milliseconds. However, the web-based client computes the moving averages with a running time that is enough to cause the human visual perception to notice flicker. The rendering superiority of the app-based client is due to implementation of the Android graphics.

#### 4.1.8 Scatter Plot Evaluation

In this section, we make a comparison of the web-based and app-based clients based on their performance in generating scatter plots, and on their performance in aggregating data points.

First of all, the web-based and app-based clients are evaluated on rendering small scatter plots: 10000, 20000, and 30000 points. The running times, measured in milliseconds, are shown in figure 37. And figure 38 shows the speedup gained by the

web-based client. The two clients perform roughly equal, with the web-based client having a slight advantage.

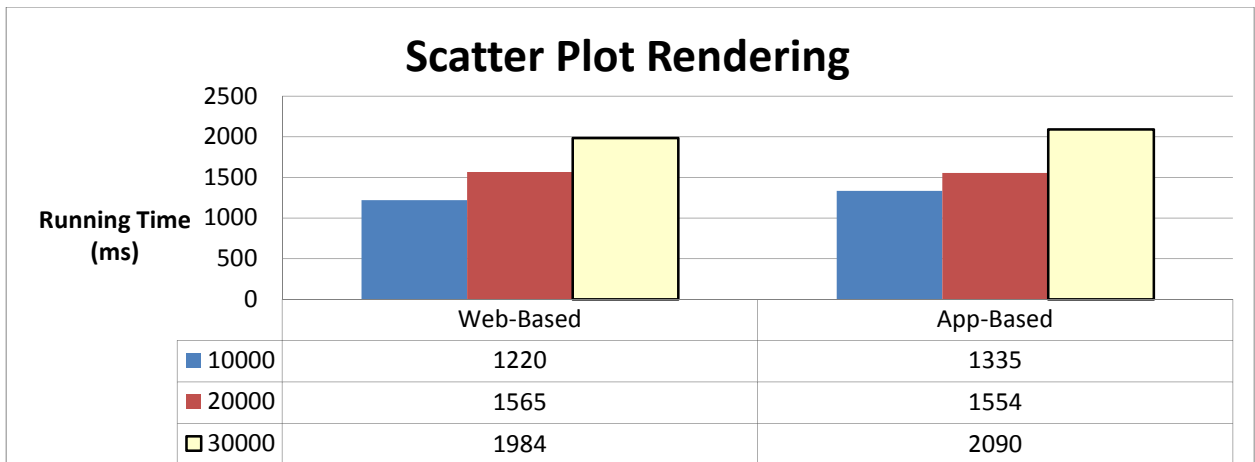


Figure 37. Scatter plot rendering

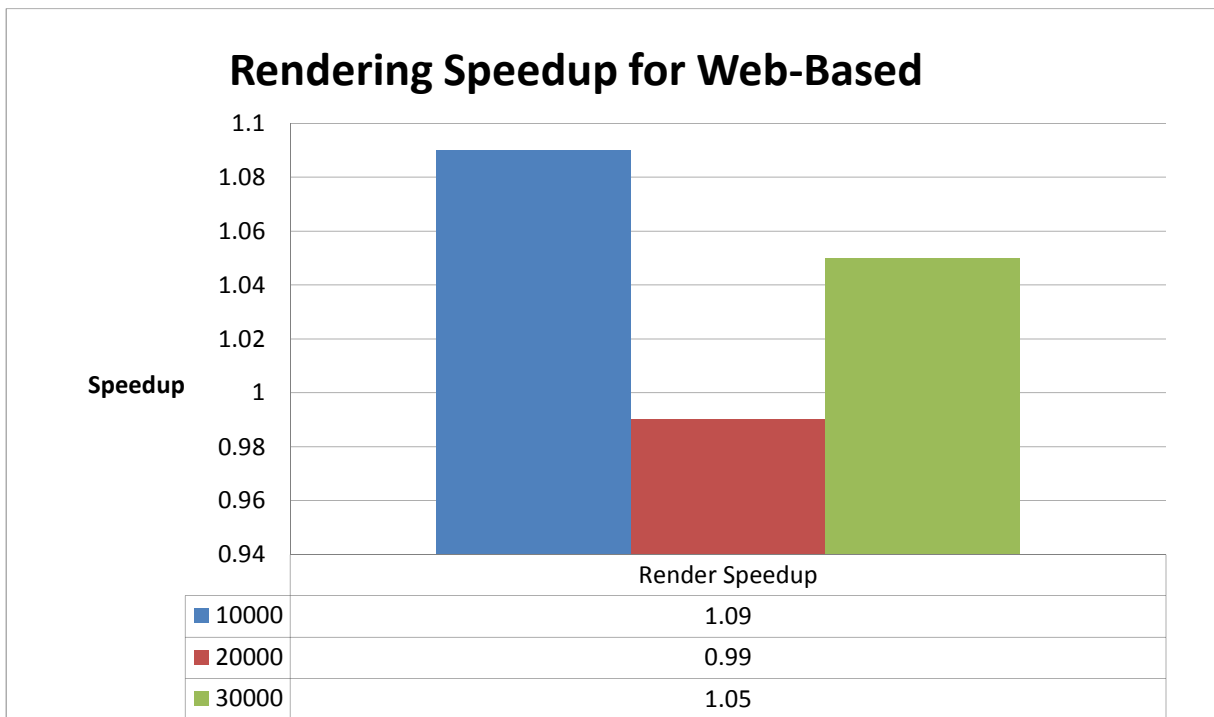


Figure 38. Scatter plot rendering speedup for web-based

Secondly, the web-based and app-based clients are evaluated on rendering large scatter plots: 100000, 125000, and 150000 points. The running times, measured in milliseconds, are shown in figure 39. And figure 40 shows the speedup gained by the

web-based client. The web-based client renders large scatter plots about three times faster than the app-based client. The app-based client is slower because it stores points in a dynamic data structure, and thus traversing a long array of points is a time consuming process.

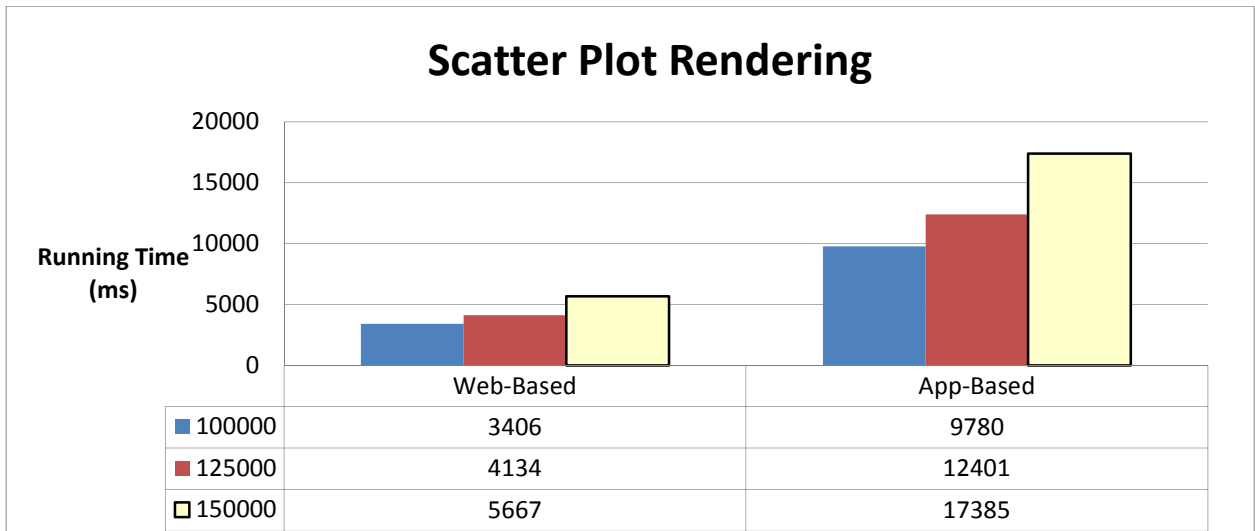


Figure 39. Large Scatter plot rendering

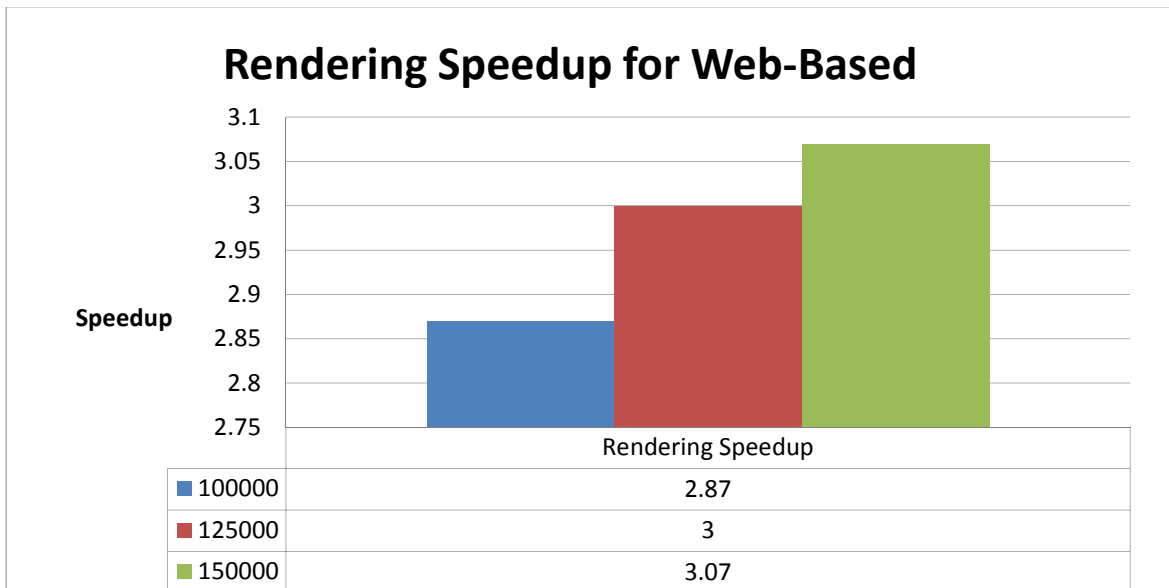


Figure 40. Large Scatter plot rendering speedup for web-based

Finally, the two clients are compared on aggregating the data points in a scatter plot. The aggregation function is triggered on every *TouchMove* event to allow the user to

observe the differences of selecting neighboring points without making repetitive touch selections. As a result, the user may generate a large number of aggregations in a short period of time. Therefore, aggregation performance is important to the human user.

To evaluate the two clients on scatter plot aggregation, we use six sub-cubes from the *Adventure Works* data cube. The first sub-cube is composed of the dimensions [Reseller].[BankName] and [Date].[DayOfYear]. The second sub-cube is [Reseller].[OrderMonth] and [Date].[DayOfYear]. The third sub-cube is [Empolyee].[DepartmentName] and [Date].[DayOfYear]. The fourth sub-cube is [Empolyee].[Empolyee] and [Date].[DayOfYear]. The fifth sub-cube is [Empolyee].[EmpolyeeDepartment] and [Date].[DayOfYear]. The sixth sub-cube is [Geography].[Geography] and [Date].[DayOfYear]. The detailed information of each sub-cube is shown in table 5. The two measures for the x and y axis are [Internet Sales Amount] and [Internet Order Quantity].

|             | Tuples | Points | Download Time (ms) | Web-Based Parse Time (ms) | App-Based Parse Time (ms) |
|-------------|--------|--------|--------------------|---------------------------|---------------------------|
| Sub-Cube #1 | 5110   | 2555   | 839                | 29                        | 9                         |
| Sub-Cube #2 | 9490   | 4745   | 1044               | 46                        | 11                        |
| Sub-Cube #3 | 11680  | 5840   | 1156               | 40                        | 11                        |
| Sub-Cube #4 | 181770 | 90885  | 8288               | 326                       | 241                       |
| Sub-Cube #5 | 216080 | 108040 | 10074              | 497                       | 132                       |
| Sub-Cube #6 | 478150 | 239075 | 21879              | 849                       | 275                       |

Table 5. Sub-Cube information for scatter plot aggregation



Figure 41 shows the running times of the clients, and figure 42 shows the speedup gained by the app-based client. The app-based client performs the aggregation function about four times faster than the web-based client, which results in smoother data visualization. This is due to Java performing calculations faster than JavaScript.

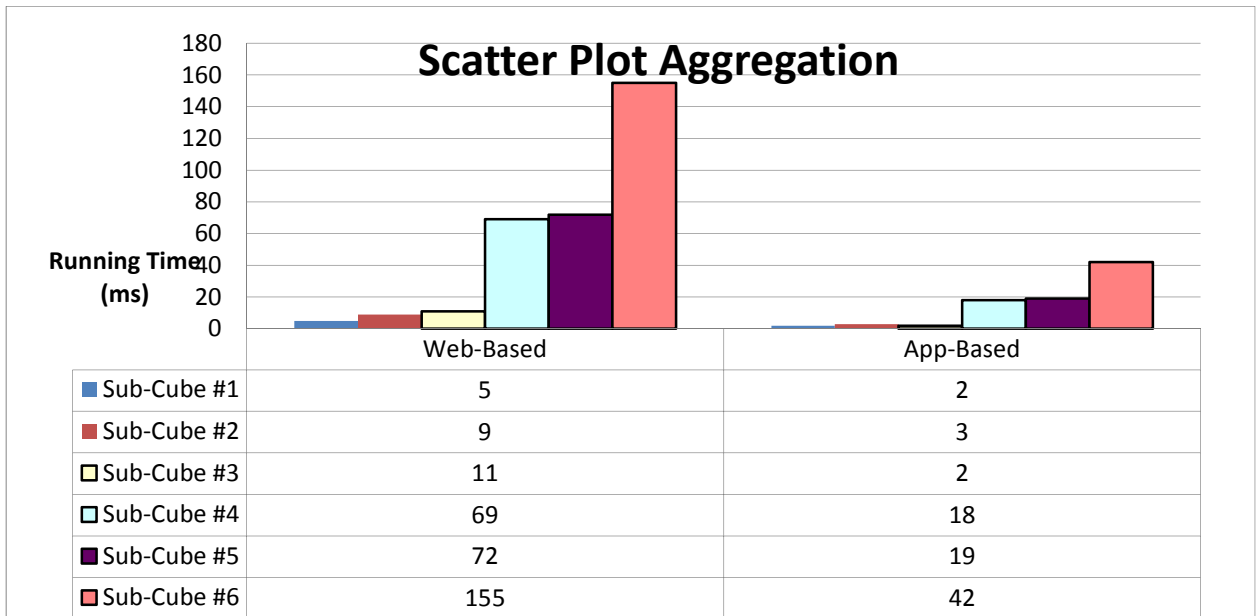


Figure 41. Scatter plot aggregation

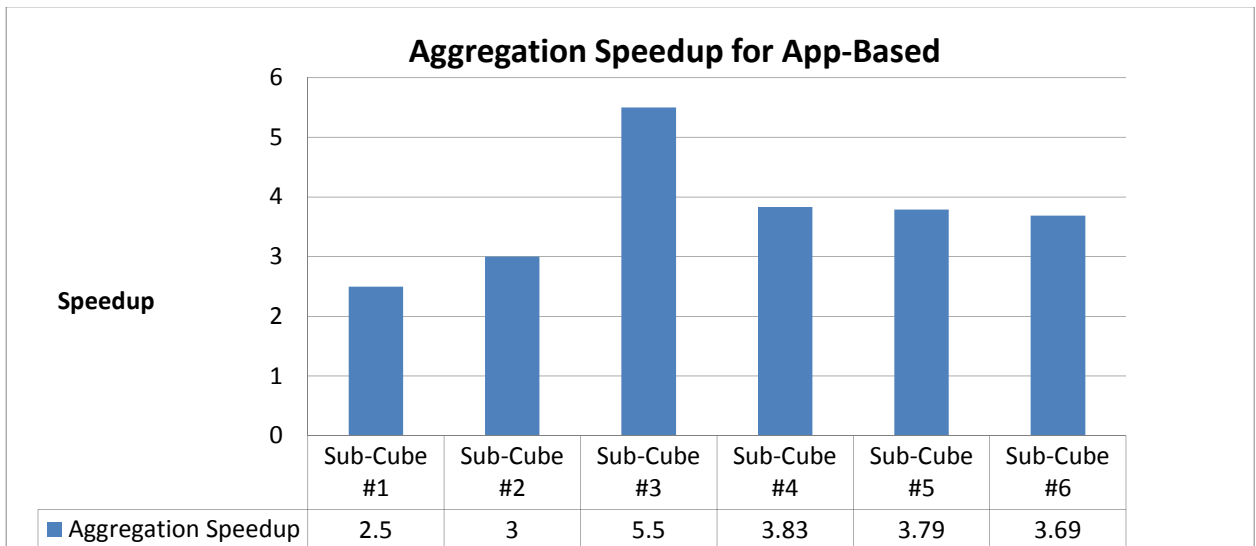


Figure 42. Scatter plot aggregation speedup

## 4.2 Server-Centric Performance

The server-centric architecture has an application server that handles all computations requested by the thin client. The thin client is merely a graphical user interface that displays the results the application server sends.

The goal of this section is to compare the performance of the client-centric architecture with a server-centric architecture and show that performing successive calculations would create jitter for a server-centric system. The clients are app-based in both architectures and use the same graphics rendering library, aChartEngine [23]. In addition, we use the same sub-cubes as chapter 4.1. We evaluate the two systems on performing top-k and moving average. In the traditional server-centric architecture, the scatter plot is rendered on the server side and is sent to the thin client as a static image. As a result, the server-centric architecture does not allow the analyst to perform aggregations on a region in the plot. Therefore in this chapter, we do not evaluate the two architectures on performing a scatter plot aggregation.

The mobile thin client for the evaluation is subjected to two real-life network environments. The first location has higher network latency, while the second location has lower network latency.

### 4.2.1 Top-k Comparison with Client-Centric Architecture

This section compares the two architectures on performing the top-k percentage contribution. Since every movement of the slider triggers a function on the application server to re-computes the top-k percentage contribution, the number of queries is quite large. Keeping up with the user would be difficult for the server-centric architecture in the presence of network latency.

The response times, measured in milliseconds, are in figure 43. Figure 44 shows the speedup gained by the client-centric architecture. The network latency introduces an additional overhead to the response time, preventing the mobile client from rendering the visualization seamlessly without jitter. For the first location, the app-based client-centric system is almost twenty times faster than the server-centric system.

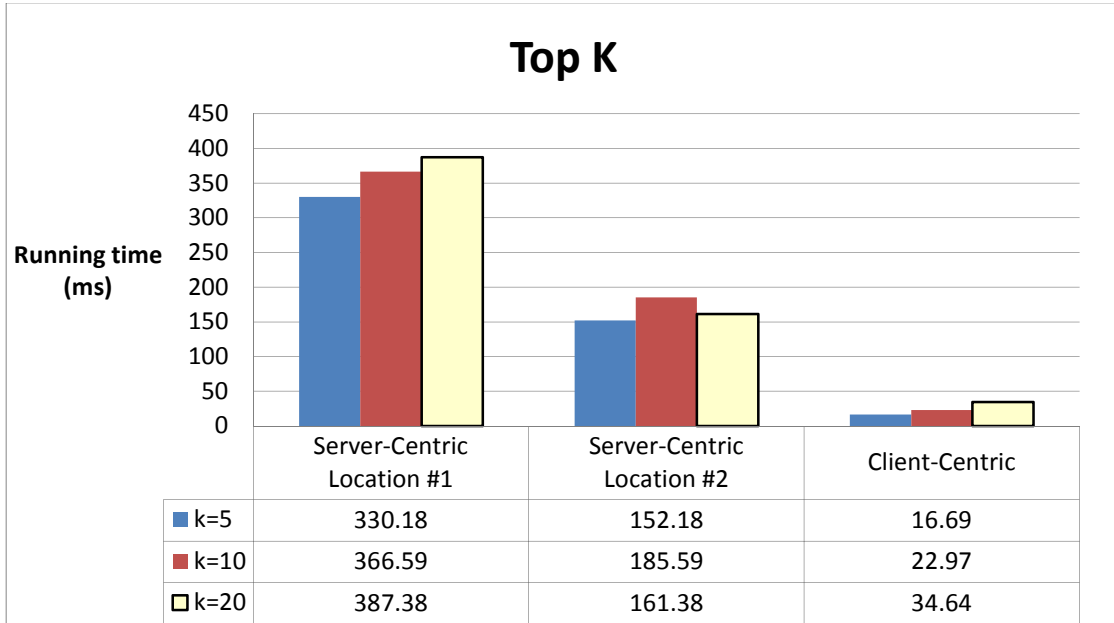


Figure 43. Top-k server-centric compared with client-centric

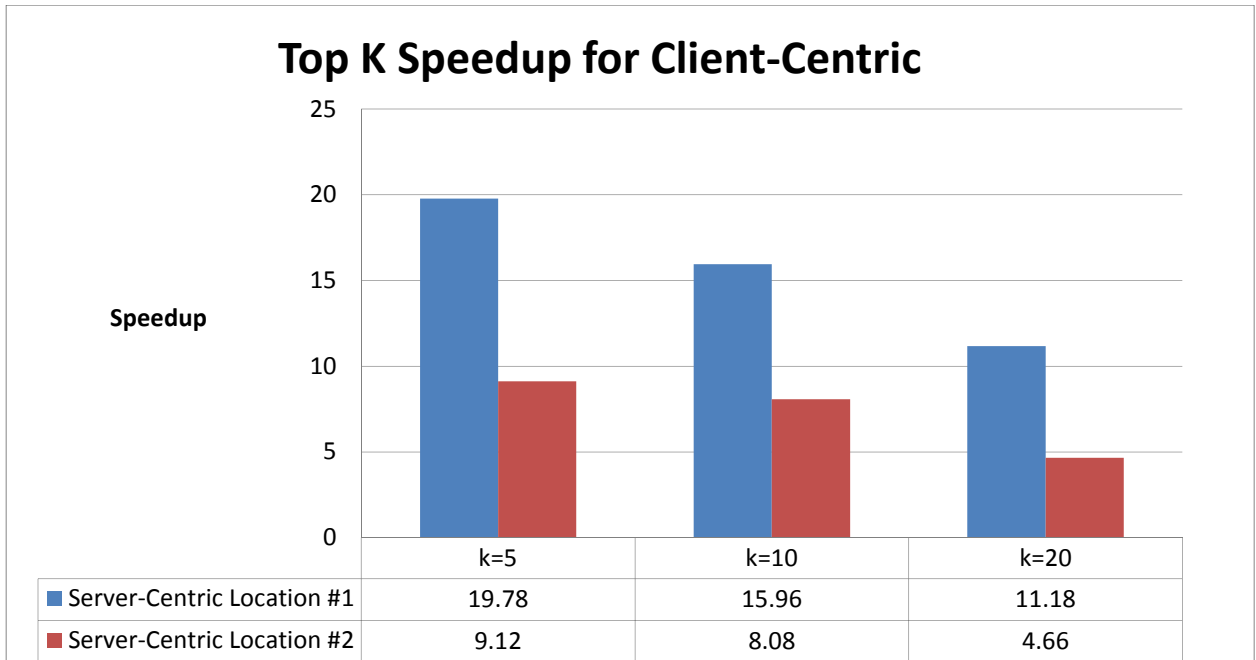


Figure 44. Top-k speedup for client-centric

#### 4.2.2 Moving Average Comparison with Client-Centric Architecture

This section compares the two architectures on performing moving averages. In the server-centric architecture, when the user starts the play back function, the thin client requests the application server for the moving average of the next day. The additional network latency prevents the thin client from updating the visualization seamlessly, in contrast to the client-centric architecture. The response times, measured in milliseconds, are in figure 45. The speedup for the client-centric architecture is shown in figure 46. For the first location, the client-centric architecture is greatly superior to the server-centric architecture with a speedup as high as fourteen. Therefore, it is clear that the client-centric architecture provides a much better user experience.

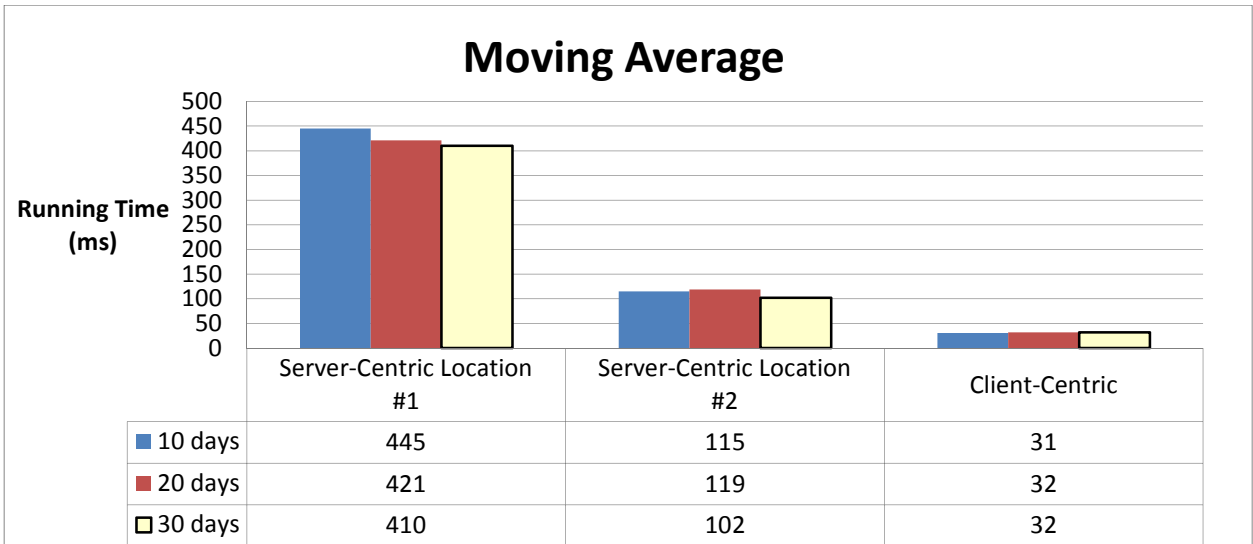


Figure 45. Moving average server-centric compared with client-centric

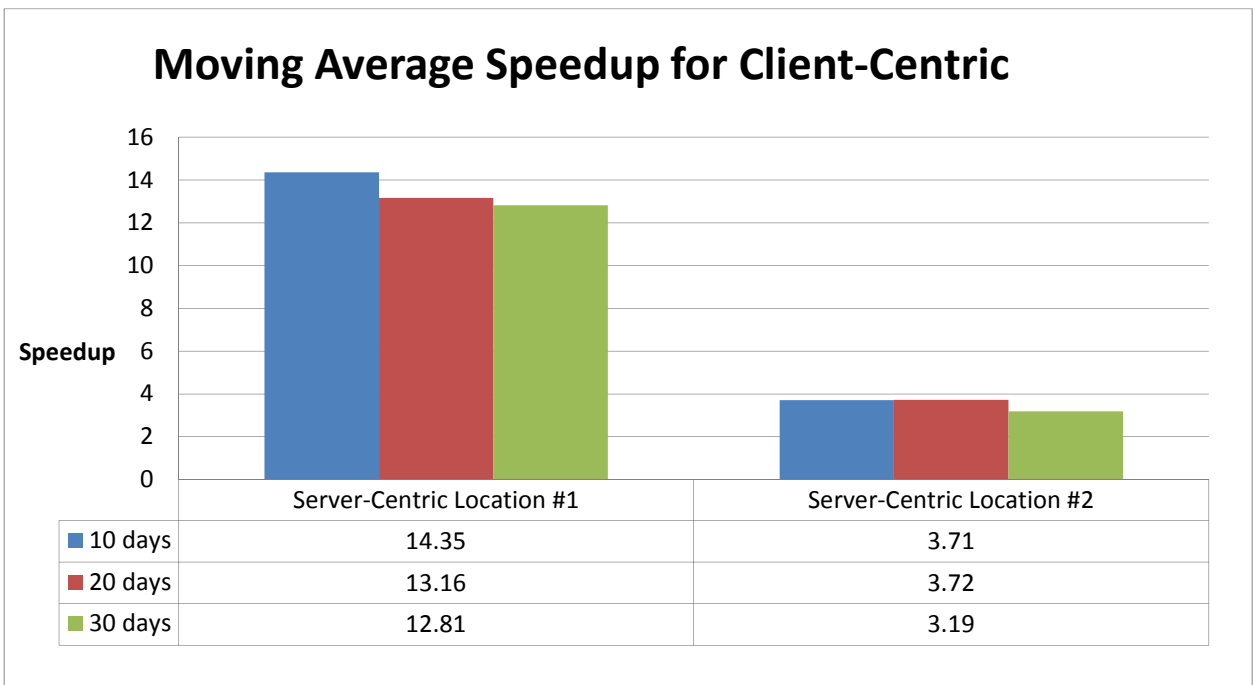


Figure 46. Moving average speedup for client-centric

## 5. CONCLUSION AND FUTURE WORK

### 5.1 Conclusion

With the emergence of powerful, multi-core mobile devices, OLAP applications are being developed for mobile software platforms. This project investigated and evaluated three OLAP architectures and environments for mobile devices: client-centric web-based, client-centric app-based and server-centric app-based.

First of all, we implemented an app-based OLAP client, and compared its performance with a modified web-based client. Although the web-based client is more convenient to use (it is not installed in the OS), it suffers from a performance penalty. Through experimentation, the web-based client was demonstrated to be inferior to the app-based client not only in performing a basic OLAP operation, but also in performing more complex, graphical visualizations, with a few exceptions. The app-based client computes a cross-tab in serial about two times faster than the web-based client, computes top-k percentage contributions two to three times faster, computes the moving averages about five times faster, and performs scatter plot aggregations about four times faster. In addition, the app-based client parses the metadata and sub-cubes downloaded from the web services faster than the web-based client. The few exceptions are rendering the data grid and the scatter plot, by which the web-based client is superior to the app-based client.

Secondly, we demonstrated the capability of the app-based client to compute a cross-tab on a recently introduced multi-core tablet. In contrast, we find it difficult for the web-based OLAP client to take advantage of the multi-core CPU. Since the web-based client lacks the software infrastructure to handle data parallelism, the app-based client

using parallel computing is up to 4.81 times faster than the web-based client in performing cross-tabs.

In addition, we found out from our experience that the performance of the app-based system has more room for improvement than the web-based one. Initially, the parsing of the metadata and sub-cube for the app-based client is substantially slower than that for the web-based client. By switching to a different parsing method, the reverse is true. On the other hand, any substantial improvements in performance of web-based systems will depend on future revisions of HTML 5.

Finally, we used two graphical visualizations to argue for the superiority of the client-centric architecture over the server-centric architecture in performance. In contrast to previous research, we used two real-life network environments, instead of using simulated environments. The additional network latency prevents the thin client from updating the visualization seamlessly, in contrast to mobile client in the client-centric architecture.

## **5.2 Future Work**

Improvements could be made to the web-based client. The HTML-based graphical libraries were proven to be inadequate in performing data visualizations relative to the app-based client. Perhaps performance could be improved if web-based client was implemented using Silverlight, which has hardware accelerated graphical libraries.

In addition, we could investigate how much improvement in scalability the client-centric architecture provides to mobile OLAP.

## REFERENCES

- [1] Alfredo Cuzzocrea, Domenico Saccà. “Hand-OLAP: Semantics-Aware Compression of Data Cubes for Effective and Efficient OLAP in Mobile Enviroments”. In *Mobile Data Management* (1) 2011: 345-348
- [2] Wayne Eckerson. “Architecting for Mobile BI”. Website, 2011  
[http://www.b-eye-network.com/blogs/eckerson/archives/2011/04/architecting\\_fo.php](http://www.b-eye-network.com/blogs/eckerson/archives/2011/04/architecting_fo.php)
- [3] Z.Wang, F.Lin, L.Zhong,and M.Chishtie. “Why are Web Browsers Slow on Smartphones?”In *Proc. ACM Int. Workshop on Mobile Computing Systems and Applications (HotMobile)*, March 2011.
- [4] The NPD Group. “As Smartphone Prices Fall, Retailers Are Leaving Money on the Table”. Website, 2011. <http://www.prweb.com/releases/2011/11/prweb8960674.htm>
- [5] RTTNews. “Tablets Gear Up to Replace PCs”. Website, 2012  
<http://www.nasdaq.com/article/tablets-gear-up-to-replace-pcs-20120323-00384>
- [6] J.Huang, Q.Xu, B.Tiwana, Z.Mao, M.Zhang, P.Bahl. “Anatomizing Application Performance Differences on Smartphones”. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, June 15-18, 2010, San Francisco, California, USA
- [7] Stanford Visualization Group. “Protovis”. Website, 2012  
<http://mbostock.github.com/protovis/>
- [8] ASUS. “ASUS Transformer Prime”. Website, 2012.  
<http://eee.asus.com/en/eeepad/transformer-prime/features/>
- [9] Wikipedia. “Android”. Website, 2012.  
[http://en.wikipedia.org/wiki/Android\\_\(operating\\_system\)](http://en.wikipedia.org/wiki/Android_(operating_system))
- [10] E.Sperling. “Rise of the Tablet Computer”. Website, 2012.  
<http://www.forbes.com/2010/09/10/ipad-apple-blackberry-technology-cio-network-tablets.html>
- [11] B.Worthen. “Businesses Add iPads to Their Briefcases”. Website, 2010.  
[http://online.wsj.com/article/SB10001424052748703846604575447531699309858.html?mod=WSJ\\_hps\\_MIDDLESixthNews](http://online.wsj.com/article/SB10001424052748703846604575447531699309858.html?mod=WSJ_hps_MIDDLESixthNews)
- [12] R.King. “MicroStrategy’s Corporate Apps Boost Productivity”. Website, 2010.  
[http://www.businessweek.com/technology/content/nov2010/tc2010111\\_549315.htm](http://www.businessweek.com/technology/content/nov2010/tc2010111_549315.htm)
- [13] SAP BusinessObjects. “SAP BusinessObjects Mobile App”. Website, 2012.  
<http://www.sap.com/solutions/sapbusinessobjects/large/business-intelligence/mobile/sap-businessobjects-mobile/index.epx>



- [14] J.Zhang. "OLAP DATABASE COMPUTATION WITH A SPLITCUBE IN A CLUSTER". Master Thesis. School of Computing Science, Simon Fraser University, 2009
- [15] L.Jiang. "OLAP DATABASE COMPUTATION WITH A SPLITCUBE ON A MULTICORE SYSTEM". Master Project Report. School of Computing Science, Simon Fraser University, 2009
- [16] F.Dehne, et al. "Parallelizing the Data Cube". In Journal of Distributed and Parallel Databases- Special Issue: Parallel and distributed data mining. Volume 11 Issue 2, March 2002.
- [17] Arnaud Giacometti , Patrick Marcel , Elsa Negre, "A framework for recommending OLAP queries". In Proceeding of the ACM 11th international workshop on Data warehousing and OLAP, October 30-30, 2008, Napa Valley, California, USA
- [18] Carl Nolan. "Manipulate and Query OLAP Data Using ADOMD and Multidimensional Expressions". Website, 1999. <http://www.microsoft.com/msj/0899/mdx/mdx.aspx>
- [19] "What is XML for Analysis (XMLA)?" Website, 2012. <http://news.xmlforanalysis.com/what-is-xmla.html>
- [20] "XML for Analysis Specification". Website, 2001. [http://msdn.microsoft.com/en-us/library/ms977626.aspx#xmlanalysis\\_topic4](http://msdn.microsoft.com/en-us/library/ms977626.aspx#xmlanalysis_topic4)
- [21] S. Chaudhuri and U.Dayal, "An overview of data warehousing and OLAP technology". In SIGMOD Record, vol.26,pp.65-74,1997
- [22] kSOAP2 team, "kSOAP2". Website, 2012. <http://ksoap2.sourceforge.net/>
- [23] AChartEngine team, "AChartEngine". Website, 2012. <http://www.achartengine.org/>
- [24] M.Schrader , et all. "Oracle Essbase & Oracle OLAP". In McGraw-Hill, Inc., New York, NY, 2009
- [25] S.Ji, B.Lee, et all. "A Study on the Generation of OLAP Data Cube Based on 3D Visualization Interaction". In Proceeding ICCSA '11 Proceedings of the 2011 International Conference on Computational Science and Its Applications Pages 231-234.
- [26] A.Maniatis , P.Vassiliadis , S.Skiadopoulos , Y.Vassiliou. "Advanced visualization for OLAP". In Proceedings of the 6th ACM international workshop on Data warehousing and OLAP, November 07-07, 2003, New Orleans, Louisiana, USA
- [27] T.Hsiao. "Web-based OLAP". Master Thesis. School of Computing Science, Simon Fraser University
- [28] Wikipedia. "Android System Architecture". Website, 2012. <http://en.wikipedia.org/wiki/File:Android-System-Architecture.svg>

- [29] E. Fortuna, O. Anderson, L. Ceze, and S. Eggers. "A limit study of JavaScript parallelism". In IEEE Intl. Symposium on Workload Characterization (IISWC), 2010, pp. 1-10.
- [30] Mozilla Developer Network. "Using web workers". Website, 2012. [https://developer.mozilla.org/En/Using\\_web\\_workers#Advanced\\_passing\\_JSON\\_Data\\_and\\_creating\\_a\\_switching\\_system](https://developer.mozilla.org/En/Using_web_workers#Advanced_passing_JSON_Data_and_creating_a_switching_system)
- [31] Intel Lab. "River Trail". Website, 2012. <https://github.com/RiverTrail/RiverTrail/wiki>
- [32] WebCL working group. "WebCL-Heterogeneous parallel computing in HTML5 web browsers". Website, 2012. <http://www.khronos.org/webcl/>
- [33] Google Android. "TextView". Website, 2012. <http://developer.android.com/reference/android/widget/TextView.html>
- [34] Google Developers. "Google Chart Tools." Website, 2012. <https://developers.google.com/chart/>
- [35] O.Laursen. "flot". Website, 2012. <http://code.google.com/p/flot/>
- [36] IBM. "Cognos software". Website, 2012. <http://www-01.ibm.com/software/analytics/cognos/>