An Inside Look at Google BigQuery

Table of Contents

Google™

An Inside Look at Google BigQuery

by Kazunori Sato, Solutions Architect, Cloud Solutions team

Abstract

This white paper introduces Google BigQuery, a fully-managed and cloud-based interactive query service for massive datasets. BigQuery is the external implementation of one of the company's core technologies whose code name is Dremel. This paper discusses the uniqueness of the technology as a cloud-enabled massively parallel query engine, the differences between BigQuery and Dremel, and how BigQuery compares with other technologies such as MapReduce/Hadoop and existing data warehouse solutions.

How Google Handles Big Data Daily Operations

Google handles Big Data every second of every day to provide services like Search, YouTube, Gmail and Google Docs.

Can you imagine how Google handles this kind of Big Data during daily operations? Just to give you an idea, consider the following scenarios:

- What if a director suddenly asks, "Hey, can you give me yesterday's number of impressions for AdWords display ads – but only in the Tokyo region?".
- Or, "Can you quickly draw a graph of AdWords traffic trends for this particular region and for this specific time interval in a day?"

What kind of technology would you use to scan Big Data at blazing speeds so you could answer the director's questions within a few minutes? If you worked at Google, the answer would be Dremel[1].

Dremel is a query service that allows you to run SQL-like queries against very, very large data sets and get accurate results in mere seconds. You just need a basic knowledge of SQL to query extremely large datasets in an ad hoc manner. At Google, engineers and non-engineers alike, including analysts, tech support staff and technical account managers, use this technology many times a day.

BigQuery: Externalization of Dremel

Before diving into Dremel, we should briefly clarify the difference between Dremel and **Google BigQuery**. BigQuery is the public implementation of Dremel that was recently launched to general availability. BigQuery provides the core set of features available in Dremel to third party developers. It does so via a REST API, a command line interface, a Web UI, access control and more, while maintaining the unprecedented query performance of Dremel.

In this paper, we will be discussing Dremel's underlying technology, and then compare its externalization, BigQuery, with other existing technologies like MapReduce, Hadoop and Data Warehouse solutions.

Dremel Can Scan 35 Billion Rows Without an Index in Tens of Seconds

Dremel, **the cloud-powered massively parallel query service**, shares Google's infrastructure, so it can parallelize each query and run it on **tens of thousands of servers** simultaneously. You can see the economies of scale inherent in Dremel.

Google's Cloud Platform makes it possible to realize super fast query performance at very attractive cost-to-value ratio. In addition, there's no capital expenditure required on the user's part for the supporting infrastructure.

As an example, let's consider the following SQL query, which requests the Wikipedia® content titles that includes numeric characters in it:

```
select count(*) from publicdata:samples.wikipedia where REGEXP_MATCH (title, '[0-9]*') AND wp_namespace = 0;
```

Notice the following:

- This "wikipedia" table holds all the change history records on Wikipedia's article content and consists of **314 millions of rows – that's 35.7GB**.
- The expression **REGEXP_MATCH(title, '[0-9]+')** means it executes a regular expression matching on title of each change history record to extract rows that includes numeric characters in its title (e.g. "List of top 500 Major League Baseball home run hitters" or "United States presidential election, 2008").
- Most importantly, note that there was **no index** or any pre-aggregated values for this table prepared in advance.

When you issue the query above on BigQuery, you get the following results with an interactive response time of **10 seconds** in most cases.

```
223,163,387
```

Here, you can see that there are about 223 million rows of Wikipedia change histories that have numeric characters in the title. This result was aggregated by actually applying regular expression matching on all the rows in the table as a **full scan**.

Dremel can even execute a complex **regular expression text matching** on a huge logging table that consists of about **35 billion rows and 20 TB, in merely tens of seconds**. This is the power of Dremel; **it has super high scalability** and most of the time it returns results within seconds or tens of seconds no matter how big the queried dataset is.

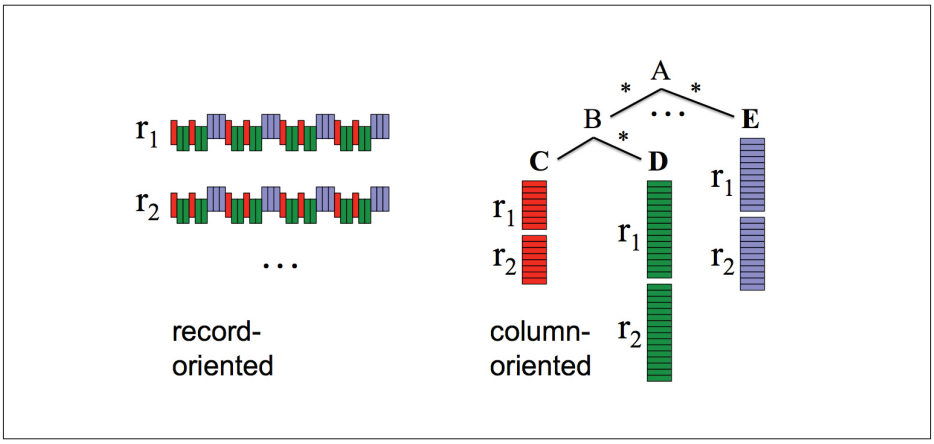
Columnar Storage and Tree Architecture of Dremel

Why Dremel can be so drastically fast as the examples show? The answer can be found in two core technologies which gives Dremel this unprecedented performance:

1. **Columnar Storage**. Data is stored in a **columnar storage** fashion which makes possible to achieve very high compression ratio and scan throughput.
2. **Tree Architecture** is used for dispatching queries and aggregating results across thousands of machines in a few seconds.

Columnar Storage

Dremel stores data in its columnar storage, which means it separates a record into column values and stores each value on different storage volume, whereas traditional databases normally store the whole record on one volume.



Columnar storage of Dremel

This technique is called Columnar storage and has been used in traditional data warehouse solutions. Columnar storage has the following advantages:

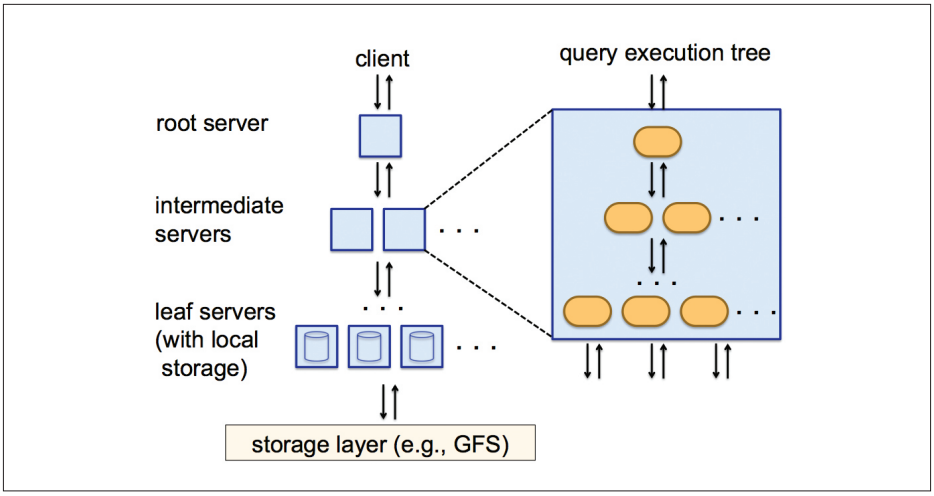
- **Traffic minimization**. Only required column values on each query are scanned and transferred on query execution. For example, a query "**SELECT top(title) FROM foo**" would access the title column values only. In case of the Wikipedia table example, the query would scan only 9.13GB out of 35.7GB.
- **Higher compression ratio**. One study[3] reports that columnar storage can achieve a compression ratio of 1:10, whereas ordinary row-based storage can compress at roughly 1:3. Because each column would have similar values, especially if the cardinality of the column (variation of possible column values) is low, it's easier to gain higher compression ratios than row-based storage.

Columnar storage has the disadvantage of not working efficiently when updating existing records. In the case of Dremel, it simply doesn't support any update operations. Thus the technique has been used mainly in read-only OLAP/BI type of usage.

Although the technology has been popular as a data warehouse database design, Dremel is one of the first implementations of a columnar storage-based analytics system that harnesses the computing power of many thousands of servers and is delivered as a cloud service.

Tree Architecture

One of the challenges Google had in designing Dremel was how to dispatch queries and collect results across tens of thousands of machines in a matter of seconds. The challenge was resolved by using the Tree architecture. The architecture forms a massively parallel distributed tree for pushing down a query to the tree and then aggregating the results from the leaves at a blazingly fast speed.



Tree architecture of Dremel

By leveraging this architecture, Google was able to implement the distributed design for Dremel and realize the vision of the massively parallel columnar-based database on the cloud platform.

These previous technologies are the reason of the breakthrough of Dremel's unparalleled performance and cost advantage.

For technical details on columnar storage and tree architecture of Dremel, refer to the Dremel paper[1] .

Dremel: Key to Run Business at "Google Speed"
Google has been using Dremel in production since 2006 and has been continuously evolving it for the last 6 years. Examples of applications include[1]:

- Analysis of crawled web documents
- Tracking install data for applications in the Android Market
- Crash reporting for Google products
- OCR results from Google Books
- Spam analysis
- Debugging of map tiles on Google Maps
- Tablet migrations in managed Bigtable instances
- Results of tests run on Google's distributed build system
- Disk I/O statistics for hundreds of thousands of disks
- Resource monitoring for jobs run in Google's data centers
- Symbols and dependencies in Google's codebase

As you can see from the list, Dremel has been an important core technology for Google, enabling virtually every part of the company to operate at "Google speed" with Big Data.

And what is BigQuery?
Google recently released BigQuery as a publicly available service for any business or developer to use. This release made it possible for those outside of Google to utilize the power of Dremel for their Big Data processing requirements.

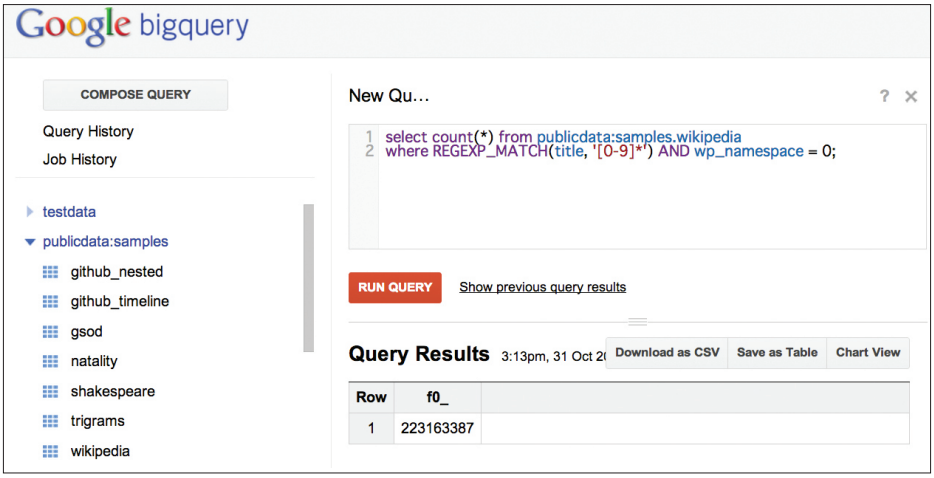


Figure 1 Querying Sample Wikipedia Table on BigQuery
(You can try out BigQuery by simply sign up for it.)

BigQuery provides the core set of features available in Dremel to third party developers. It does so via a REST API, command line interface, Web UI, access control, data schema management and the integration with Google Cloud Storage.

BigQuery and Dremel share the same underlying architecture and performance characteristics. Users can fully utilize the power of Dremel by using BigQuery to take advantage of Google's massive computational infrastructure. This incorporates valuable benefits like multiple replication across regions and high data center scalability. Most importantly, this infrastructure requires no management by the developer.

BigQuery versus MapReduce

In the following sections, we will discuss how BigQuery compares to existing Big Data technologies like MapReduce and data warehouse solutions.

Google has been using MapReduce for Big Data processing for quite some time, and unveiled this in a research paper[2] in December of 2004. Some readers may have heard about this product, and its open source implementation **Hadoop**, and may wonder about the difference between the two. This is the difference:

- Dremel is designed as an **interactive** data analysis tool for large datasets
- MapReduce is designed as a programming framework to **batch process** large datasets

Moreover, Dremel is designed to finish most queries within seconds or tens of seconds and can even be used by non-programmers, whereas MapReduce takes much longer (at least minutes, and sometimes even hours or days) to finish processing a dataset query.

Comparing BigQuery and MapReduce

MapReduce is a distributed computing technology that allows you to implement custom "mapper" and "reducer" functions programmatically and run batch processes with them on hundreds or thousands of servers concurrently. The following figure shows the data flow involved. Mappers extract words from text, and reducers aggregates the counts of each word.

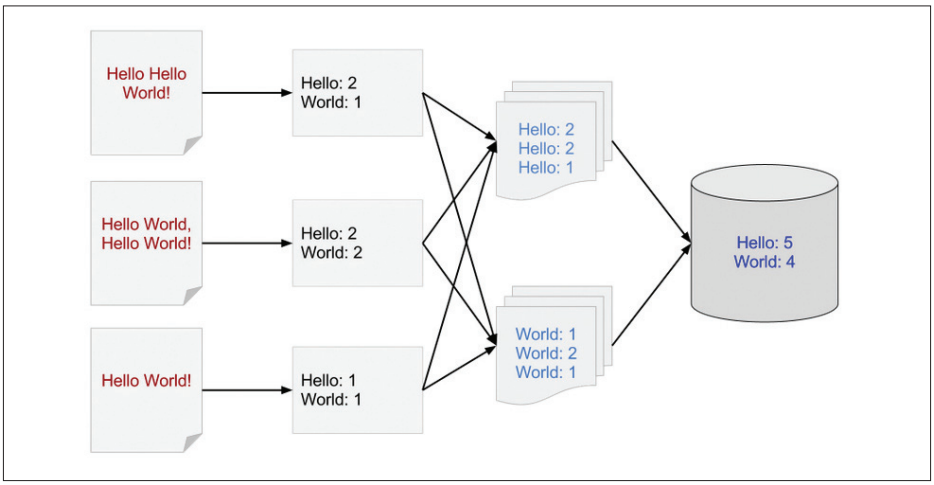


Figure 2 MapReduce Data Flow

By using MapReduce, enterprises can cost-effectively apply parallel data processing on their Big Data in a highly scalable manner, without bearing the burden of designing a large distributed computing cluster from scratch, or purchasing expensive high-end relational database solutions or appliances.

In the last several years, Hadoop, the open-source implementation of MapReduce, has been a popular technology for processing Big Data for various applications such as log analysis, user activity analysis for social apps, recommendation engines, unstructured data processing, data mining, and text mining, among others.

MapReduce Limitations

As a former AdWords API traffic analyst, I sometimes used Google's internal MapReduce frontend called Tenzing[4] (which is similar to Hive because it works as a SQL frontend for Hadoop) to execute multiple join operations across extremely large tables of ads data. The objective was to merge and filter them, under certain conditions, in order to to extract a list of ads for a group of accounts. MapReduce works well in scenarios like this, delivering results in a reasonable amount of time (such as, tens of minutes). If I had used traditional relational database technology, this same query would have taken an unreasonable amount time at a high cost, or simply it would have been impossible to perform the task at all.

However, MapReduce was only a partial solution, capable of handling about a third of my problem. I couldn't use it when I needed nearly instantaneous results because it was too slow. Even the simplest job would take several minutes to finish, and longer jobs would take a day or more. In addition, if the result was incorrect due to an error in the MapReduce code I wrote, I'd have to correct the error and restart the job all over again.

MapReduce is designed as a batch processing framework, so it's not suitable for ad hoc and trial-and-error data analysis. The turnaround time is too slow, and doesn't allow programmers to perform iterative or one-shot analysis tasks on Big Data.

Simply put, if I had only used MapReduce, I couldn't have gone home until the job was finished late at night. By using Dremel instead of MapReduce on about two-thirds of all my analytic tasks, I was able to finish the job by lunch time. And if you've ever eaten lunch at Google, you know that's a big deal.

The following figure shows a comparison of execution times between MapReduce and Dremel. As you can see, there is a difference in orders of magnitude.

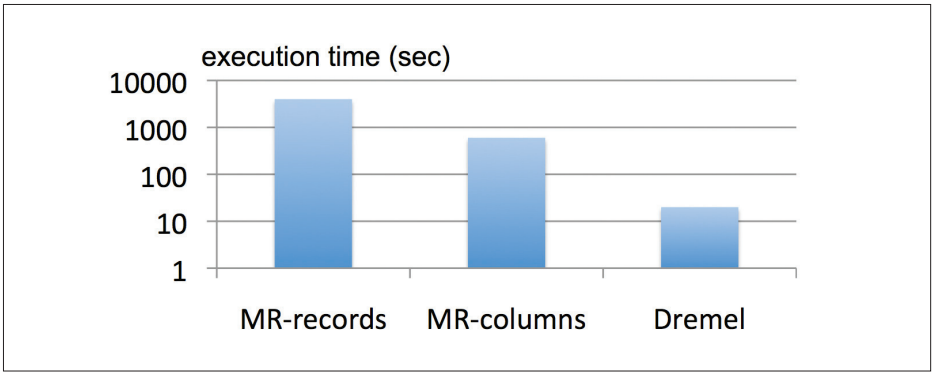


Figure 3 MapReduce and Dremel Execution Time Comparison
The comparison was done on 85 billion records and 3000 nodes. "MR-records" refers to MapReduce jobs accessing row-based storage whereas "MR-columns" refers to MR jobs with column-based storage. For more information, refer to section 7. EXPERIMENTS of the Dremel: Interactive Analysis of Web-Scale Datasets paper[1].

MapReduce and Dremel are both massively parallel computing infrastructures, but Dremel is specifically designed to run queries on Big Data in as little as a few seconds.

BigQuery and MapReduce Comparison

BigQuery and MapReduce are fundamentally different technologies and each has different use cases. The following table compares the two technologies and shows where they apply.

Key Differences	BigQuery	MapReduce
What is it?	Query service for large datasets	Programming model for processing large datasets
Common use cases	Ad hoc and trial-and- error interactive query of large dataset for quick analysis and troubleshooting	Batch processing of large dataset for time-consuming data conversion or aggregation
Sample use cases		
OLAP/BI use case	Yes	No
Data Mining use case	Partially (e.g. preflight data analysis for data mining)	Yes
Very fast response	Yes	No (takes minutes - days)
Easy to use for non-programmers (analysts, tech support, etc)	Yes	No (requires Hive/Tenzing)
Programming complex data processing logic	No	Yes
Processing unstructured data	Partially (regular expression matching on text)	Yes

Data handling		
Handling large results / Join large table	No (as of Sept 2012)	Yes
Updating existing data	No	Yes

Figure 4 MapReduce and BigQuery Comparison

BigQuery is designed to handle **structured data** using **SQL**. For example, you must to define a table in BigQuery with column definition, and then import data from a CSV (comma separated values) file into Google Cloud Storage and then into BigQuery. You also need to express your query logic in a SQL statement. Naturally, BigQuery is suitable for **OLAP** (Online Analytical Processing) or **BI** (Business Intelligence) usage, where most of the queries are simple and done through a quick aggregation and filtering by a set of columns (dimensions).

MapReduce is a better choice when you want to process **unstructured data programmatically**. The mappers and reducers can take any kind of data and apply complex logic to it. MapReduce can be used for applications such as **data mining** where you need to apply complex statistical computation or data mining algorithms to a chunk of text or binary data. And also, you may want to use MapReduce if you need to output gigabytes of data, as in the case of merging two big tables.

For example, users may want to apply these criteria to decide what technology to use:

Use BigQuery

- Finding particular records with specified conditions. For example, to find request logs with specified account ID.
- Quick aggregation of statistics with dynamically-changing conditions. For example, getting a summary of request traffic volume from the previous night for a web application and draw a graph from it.
- Trial-and-error data analysis. For example, identifying the cause of trouble and aggregating values by various conditions, including by hour, day and etc...

Use MapReduce

- Executing a complex data mining on Big Data which requires multiple iterations and paths of data processing with programmed algorithms.
- Executing large join operations across huge datasets.
- Exporting large amount of data after processing.

Of course, you can make the best use of both technologies by combining them to build a total solution. For example,

- Use MapReduce for large join operations and data conversions, then use BigQuery for quick aggregation and ad-hoc data analysis on the result dataset.
- Use BigQuery for a preflight check by quick data analysis, then write and execute MapReduce code to execute a production data processing or data mining.

Data Warehouse Solutions and Appliances for OLAP/BI

Many enterprises have been using data warehouse solutions or appliances for their OLAP/BI use cases for many years. Let's examine the advantages of using BigQuery for these traditional purposes:

In OLAP/BI, you roughly have the following three alternatives for increasing the performance of Big Data handling.

- Relational OLAP (ROLAP)
- Multidimensional OLAP (MOLAP)
- Full scan

Relational OLAP (ROLAP)

ROLAP is an OLAP solution based on relational databases (RDB). In order to make RDB faster, you always need to build **indices** before running OLAP queries. Without an index, the response will be very slow when running a query on Big Data. For this reason, you need to build indices for every possible query beforehand. In many cases, you need to build many indices to cover all the expected queries, and their size could become larger than original data. If the data is really large, sometimes the entire set of data and indices would require ever larger and more complex and expensive hardware to house it.

Multidimensional OLAP (MOLAP)

MOLAP is an OLAP solution that is designed to build **data cubes** or **data marts** based on dimensions predefined during the design phase. For example, if you are importing HTTP access logs into a MOLAP solution, you would choose dimensions such as "time of day", "requested URI" and "user agent" so that MOLAP can build a data cube featuring those dimensions and aggregated values. After that, analysts and users can quickly get results for queries such as "What was the total request count for a specified user agent, grouped by each time of the day?".

A weakness of MOLAP is that BI engineers must spend extensive time and money to design and build those data cubes or data marts before analysts can start using them. Sometimes these designs can be "brittle", with even the slightest schematic changes causing a failure that requires a new investment in the whole process.

Full-scan Speed Is the Solution

As you can see, neither ROLAP or MOLAP is suitable for ad hoc queries or trial-and-error data analysis, as you need to define all the possible queries at design or import time. In the real world, the ad hoc queries are a major part of OLAP requirement as we see in the case of a Googler's daily life: You can never imagine what kind of queries you would need in every possible situation. For these use cases, you need to increase the speed of **full scan** (or table scan), accessing all the records on disk drives without indexing or pre-aggregated values.

As we mentioned in an earlier section, **disk I/O throughput** is the key to full-scan performance. Traditional data warehouse solutions and appliances have tried to achieve better disk I/O throughput with the following technologies:

- **In-memory database or flash storage**. The most popular solution is to fill the database appliance with **memory** modules and **flash storage** (SSDs) to process Big Data. This is the best solution if you don't have any cost restrictions. Appliance products comprised of SSDs can cost **hundreds of thousands of dollars** when used to store Big Data.

- **Columnar storage**. This technology stores each record's column value in different storage volumes. This allows for higher compression ratio and disk I/O efficiency than ordinary row-based storage. Columnar storage has become a standard technology for data warehouse solutions since the 1990s; BigQuery (Dremel) fully utilizes it with better optimization.
- **Parallel disk I/O**. The last and most important factor in improving the throughput is the parallelism of disk I/O. The full-scan performance will increase linearly, in relation to the number of disk drives working in parallel. Some data warehouse appliances provide special storage units that allow you to run a query in parallel on tens or hundreds of disk drives. But again, since these appliances and storage solutions are all on-premise and proprietary hardware products, they tend to be quite expensive.

BigQuery solves the parallel disk I/O problem by utilizing the **cloud platform's economy of scale**. You would need to **run 10,000 disk drives and 5,000 processors simultaneously to execute the full scan of 1TB of data within one second**. Because Google already owns a huge number of disk drives in its own data centers, why not use them to realize this kind of massive parallelism?

BigQuery's Unique Abilities

Based on Dremel's novel approach, BigQuery provides extremely high cost-effectiveness and full-scan performance for ad hoc queries thanks to the unique combination of a massively parallel query engine.

Cloud-Powered Massively Parallel Query Service

Until now, this level of query performance – full scanning of 35B rows in tens of seconds without an index – has been achieved only by very expensive data warehouse appliances or by carefully integrated cluster of database servers equipped with full memory and flash storage.

Prior to the release of BigQuery, companies were spending **hundreds of thousands of dollars or more to effectively query this amount of data**[6].

In comparison, BigQuery's cost is drastically lower. To appreciate the difference in price, consider the Wikipedia query example we explored at the beginning of this paper. If you execute the query on BigQuery, it would cost you just **$0.32** for each query plus **$4.30** per month for Google Cloud Storage. As you can see, there's a huge cost savings with BigQuery versus traditional data warehouse solutions.

Note that BigQuery scans only the required columns for a query, not all the columns. Each query costs $0.035 per GB as of July 31, 2012. This example query requires 9.13 GB to scan, so it costs $0.035 x 9.13 GB = $0.32 per query. Refer to the BigQuery pricing table ***(https://developers.google.com/bigquery/docs/pricing)*** *for detailed price information.*

How to Import Big Data

Importing data into BigQuery is the first challenge to overcome when working with Big Data.

This is done following this two steps process:

1. Upload your data to Google Cloud Storage. Most of the time, the bottleneck will be your network bandwidth available to perform this step.
2. Import the files to BigQuery. This step can be executed by a command-line tool, Web UI or API, which can typically import roughly 100 GB within a half-hour. Refer to the document **(https://developers.google.com/bigquery/docs/developers_guide#importingatable)** for details.

Acknowledgements

I would like to thank the people who helped in writing and reviewing this white paper, including Ju-kay Kwek, Michael Manoochehri, Ryan Boyd, Hyun Moon, Chris Elliot, Ning Tang, Helen Chou, Raj Sarkar, Michael Miele, Laura Bergheim, Elizabeth Markman, Jim Caputo, Ben Bayer, Dora Hsu and Urs Hoelzle. I appreciate your contribution so much.

Once these solutions are available, it is easier to extract Big Data from a legacy database, apply transformations or clean-ups and import them to BigQuery.

Why Use the Google Cloud Platform?

The initial investment required to import data into the cloud is offset by the tremendous advantages offered by BigQuery. For example, as a fully-managed service, BigQuery requires no capacity planning, provisioning, 24x7 monitoring or operations, nor does it require manual security patch updates. You simply upload datasets to Google Cloud Storage of your account, import them into BigQuery, and let Google's experts manage the rest. This significantly reduces your total cost of ownership (TCO) for a data handling solution.

Growing datasets have become a major burden for many IT department using data warehouse and BI tools. Engineers have to worry about so many issues beyond data analysis and problem-solving. By using BigQuery, IT teams can get back to focusing on essential activities such as building queries to analyze business-critical customer and performance data.

Also, BigQuery's REST API enables you to easily build App Engine-based dashboards and mobile front-ends. You can then put meaningful data into the hands of associates wherever and whenever they need it.

Conclusion

BigQuery is a query service that allows you to run SQL-like queries against multiple terabytes of data in a matter of seconds. The technology is one of the Google's core technologies, like MapReduce and Bigtable, and has been used by Google internally for various analytic tasks since 2006. Google has launched Google BigQuery, an externalized version of Dremel. This release made it possible for developers and enterprises to utilize the power of Dremel for their Big Data processing requirements and accelerate their business at the same swift pace.

While MapReduce is suitable for long-running batch processes such as data mining, BigQuery is the best choice for ad hoc OLAP/BI queries that require results as fast as possible. BigQuery is the cloud-powered massively parallel query database that provides extremely high full-scan query performance and cost effectiveness compared to traditional data warehouse solutions and appliances.

References

1. Dremel: Interactive Analysis of Web-Scale Datasets **http://research.google.com/pubs/pub36632.html**
2. MapReduce: Simplified Data Processing on Large Clusters **http://research.google.com/archive/mapreduce.html**
3. Column-Oriented Database Systems, Stavros Harizopoulos, Daniel Abadi, Peter Boncz, VLDB 2009 Tutorial **http://cs-www.cs.yale.edu/homes/dna/talks/Column_Store_Tutorial_VLDB09.pdf**
4. Tenzing A SQL Implementation On The MapReduce Framework **http://research.google.com/pubs/pub37200.html**
5. Protocol Buffers – Google's data interchange format **http://code.google.com/p/protobuf/**
6. Price comparison for Big Data Appliance, Jean-Pierre Dijcks, Oracle Corporation **https://blogs.oracle.com/datawarehousing/entry/price_comparison_for_big_data**

Google™

WP2031-1210