HOW BIG IS BIG?

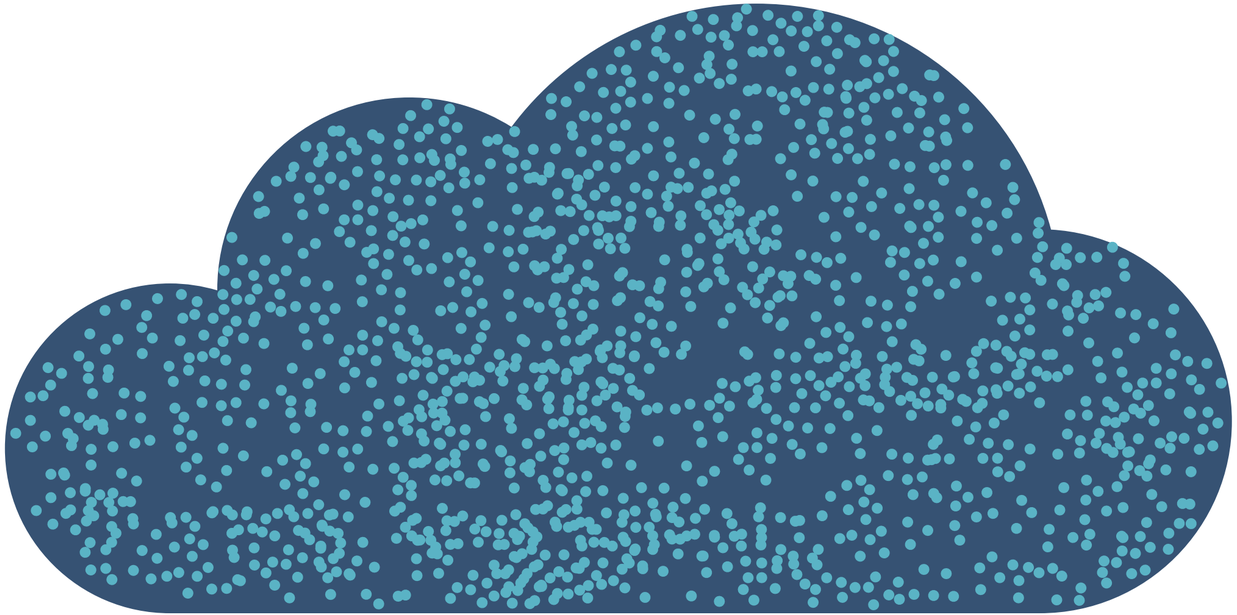
Applying Big Data concepts for analytics and visualizations at any scale

Current estimates say that by 2020 there will be more than ***40 zetabytes*** of data (about 5,200 gigabytes per person).*

While most companies will work with just a small fraction of this available data, the challenges they face in aggregating and organizing their data can benefit from approaches similar to those used in big data analysis.

Generally, big data projects follow the same process: acquire the data, align the data, store it for future use, and then shape it for analysis. Therefore, while each data aggregation project is as unique as the customer problem it addresses, there are universal questions to ask that can set you up for the best overall strategy to fit your needs.

* Source: http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf.



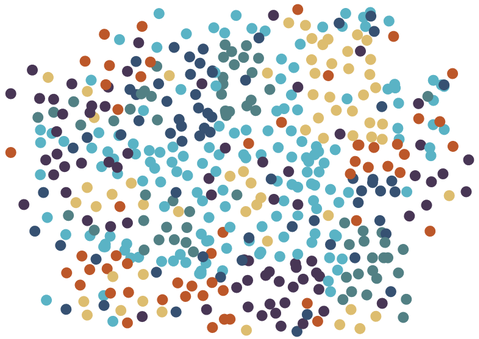
BEFORE YOU GET STARTED

The Four V's of Big Data

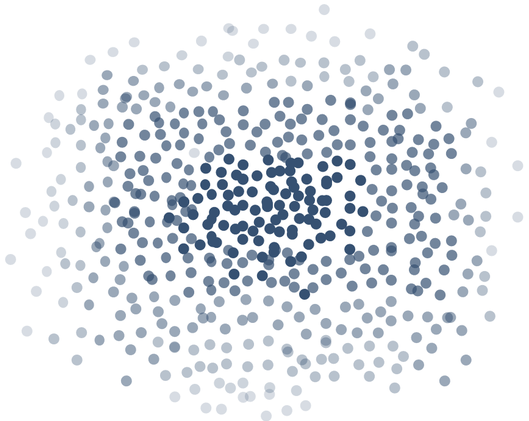
Big Data system architects commonly consider ***The Four V's*** when designing a data centric solution. Throughout this article, keep in mind how the four V's of big data apply to your situation (even if you may not have big data) as you consider each decision you make:

For more info, check out a cool infographic here: http://www.ibmbigdatahub.com/infographic/four-vs-big-data

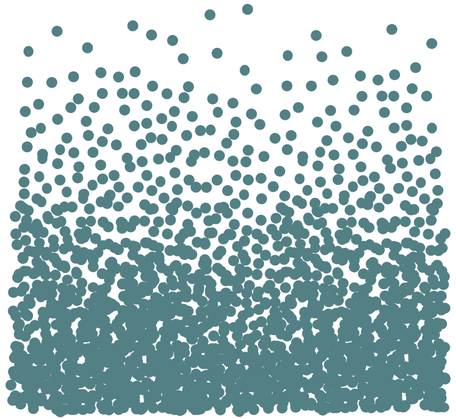
VELOCITY How fast can you get to your data? Does it differ by source?



VARIETY How many different kinds of data do you have?



VERACITY How uncertain is each piece of data? How much can it be trusted?



VOLUME How much data is there, both total and from each source individually?

1 DATA ACQUISITION AND TRANSPORTATION

How do you get your data and move it around?

The first step in data aggregation for analysis and visualization is to find all the relevant data and collect it in the same place. Typically, this involves setting up a way to acquire the data you need on a regular schedule and placing the data within the correct storage location once you have it. The best acquisition method will vary by the type of data and local storage methods that each company employs.

The following considerations will help you and your software provider narrow down which methods are right for you:

How is the data stored?

Your company probably has data in multiple locations and in many formats, so finding your data and determining its local layout is the first step in aggregation. Do you have a centralized database or multiple data sources within your company? Is some of your data kept in an Excel or .csv file that can be collected periodically? Is it streaming data from an internet connected device? Or is it stored in a local system where a web service or API could be used to collect the data?

How often do you want to acquire new data?

After you have your data collection strategy decided upon, you will need to determine how often it needs to be integrated into your analytics processes. Is it sufficient to view historic data current to the last quarter, month, week, or day, or do you need by minute to minute updates? If you are accessing your data more infrequently, then sending all the data as a batch can be done during off hours, increasing the performance of queries against the data during the day. If you need more frequent updates, then you will likely want to pursue a method of sending data in frequent small packages, or even streaming. These methods assure that you have close to real-time data, but can take a toll on performance.

How much data is being collected at the intervals you've selected?

Now you can begin to design your data workflows, but deciding what technology to use will depend on the structural characteristics of the data payloads you are moving around. Is it in large chunks (such as a batch of long distance driver logs) or in discrete pieces (like individual laboratory test results)? Most integration and data aggregation approaches leverage some type of messaging infrastructure to implement workflows. The size of your data chunks will help you decide whether to send messages that incorporate the data or to migrate the data as a chunk to temporary storage and then send messages with a location reference so that the data can be retrieved later in the process. As above, there are considerations for each choice. Messages that incorporate data tend to streamline development, but messages with location identifiers can improve performance and increase modularity of the system, making it more traceable and easier to adapt to change in the future. These considerations can also influence the tool selection process as many systems are only designed to accepted data as part of the message payload.

Can you retrieve your data again?

In many cases you need to plan for the possibility that information may be lost or corrupted during the collection and processing steps. Generally, when more traceability and modularity built into a system, it is easier to pinpoint where something went wrong and correct the issue. If your source data is stored in a file or database, then it is likely you can go back and reload the data if something unexpected happens. However, if you are collecting a live stream or a rolling log you might not be able to retrieve it again. The impact on data loss or corruption will be different depending on the methods you've chosen to acquire and move the data.

How will you monitor data flow and clean up any errors?

Like the old adage goes, "Hope for the best, but plan for the worst". Once you've made your choices on how the system should be set up, you need to know what to do when something goes wrong. Plans for downtime and the inevitable case where data is unavailable need to be put into place so that operations don't grind to a halt if your access to the data goes down. In addition, you will need to think about a data clean up strategy in the case that inaccurate or misplaced data gets into the system. In the case of inaccurate data, it is useful to know what actions were performed against the data before it got to storage. This includes what source it came from, what manipulations where done on it, and when it entered the system? It is also important to consider the possibility that decisions could be made based on the inaccurate data and determine a plan of action to prevent and correct any harm done by these events.

2 DATA ALIGNMENT

How do you make sure your data is apples to apples?

When pulling data from disparate sources it is common to find it difficult to isolate key identifiers because the data sources aren't identical. Think of a patient who goes to see their doctor, goes to a lab for a test, and then pays for it via insurance. The patient is the same in each case, but the doctor, the lab, and the insurance might use different identifiers in their database to store the information (patient id number, test result id, insurance policy number). Even in the cases where each organization keys on the same attribute (e.g. patient id number) each organization may use a different locally-sourced identifier for the patient. Because of this difference in local data storage and indexing, you typically need to normalize the keying of acquired data using techniques such as generating a common index or through heuristic matching. No matter what method you use, you want to keep the acquired data as close to the source data as possible to increase traceability later on. This allows for easier monitoring and recovery of the process in the case of errors or downtimes.

The two big questions in data alignment are: how to match up the data and how to monitor the process.

How will you match up your data?

There are many mechanisms you can use for aligning data. Here are three commonly employed methods.

Primary key: Option one is that you have a primary key that matches up each table, with or without translation. In this case, your data is already set up to be compared or you have a direct comparison table that allows you to say that the primary key for one table is equivalent to the primary key on another table. You then simply align the tables containing your data using the shared primary key.

Indexes: Another option is to use an index that allows you to connect one table to the next. This is especially handy if you don't have a natural primary key to link to. For many common types of data, there might already be a centralized index that can be used such as the common patient index or CODIS (the combined DNA indexing system). If there isn't a common index, you will need to generate one from within your system that can serve for your data.

Heuristic matching: Heuristic matching allows you to align one aggregated set of data to another through a matching criterion of your choice (temporal alignment is a frequent choice). You might choose to aggregate your data for the sake of performance or analysis, or that might be the only way the data is available to you for security purposes. Heuristic matching for data aggregation is common, for example, in healthcare where you get aggregated blinded data to protect individual patient information. It is crucial that you employ sound statistical analysis to ensure that the algorithms used in this type of matching are correct and not providing correlations that are misleading to the users of the information.

How will you monitor the alignment process?

Not only do you want to make sure you are still acquiring data, but you want to make sure that your data is still arriving in the format you expect. For this, you will want to set up a protocol for data quality assurance to check for any issues with the alignment process. For example, you will want to know if the data is coming in the expected format, if all the parameters being used in the alignment algorithm are present in each data package, etc. Often, this process can be automated, having the system send out periodic reports and exception logs. However, sometimes companies also require a manual QA process to spot check the data for issues that cannot easily be detected by automated processes and the integration approach chosen may need to incorporate this manual step.

3 DATA STORAGE AND SCALABILITY

How will you keep your data accessible for analysis now and in the future?

Once you have divined your data sources and know how they are to be related with one another, you'll want to extract portions of that information to a common platform in order to be able to access it for analytical purposes such as predictive studies or KPI dashboards.

The appropriate storage platform will depend on multiple factors, but two overarching questions to ask at each point are how *durable* and how *scalable* do you need it to be? Durability can be thought of in terms of system availability and security. If you didn't have access to your analytics system for an hour, a day or several days how does that affect your operational situation? Scalability is a measure of both how many people need access to the data and how much data you plan to accumulate and maintain operationally at any time.

To determine the best storage strategy for a given use case, consider the following options:

Cloud vs. Local storage

Cloud storage can be an attractive option for those that do not want to set up data storage operations in house and are typically scalable in a very affordable way. They also tend to be durable from the standpoint that by replicating data and storing it in separate locations you can potentially decrease downtime at a very reasonable cost. However, you have less control over your data from the standpoint that if the storage company experiences a downtime, so will you. Conversely, storing data locally can provide you with direct control over its accessibility and performance tuning. Local storage can also be highly scalable, but it makes you responsible for maintenance and security and setting up environments that can scale locally can be quite expensive.

OLAP Structured vs. Unstructured Storage

OLAP structured storage has been traditionally the most commonly used method for designing data warehousing solutions, each bit of operational information gets broken out into containers organized by a business domain hierarchy and then these containers are related to each other using dimensions such as age, gender, or date range. Traditional data warehouse requires shaping data before storage, because each bit of data has a defined location within the structure. Because of this, data is not stored in its source form, but it is readily and easily available in the correct format for analysis later on. However, in order to introduce new data types to the system, the database will need to be updated to receive these types of data.

Unstructured storage mechanisms such as NoSQL or document databases, distributed file systems like Hadoop, and highly scalable raw file or BLOB (binary large object) storage have recently been leveraged to accommodate situations where locking the inputs to one single data structure over time is untenable. Unstructured storage simply stores data as it comes, often using an indexing scheme to speed retrieval. No overarching structure is imposed on the data during the storage process and the hierarchy and attributes of the data are "discovered" during the analysis process using technologies focused on working with large amounts of potentially disparate data. Unstructured frameworks are generally used for modern, highly available systems (such as Twitter or Facebook) where companies want clients to have continual access to past data while they are updating their infrastructure. The clear upsides to unstructured storage are the ability to store data in the format in which it is received, the ability change data structures without downtimes, and the ability to design your current applications without the need to accommodate data changes in the future. The downside of this lack of structure is that you have to build the database rules into the software applications that are accessing the data and complete any translations of format on the fly as you are using the information. Unstructured data storage by its very nature does not enforce data consistency, and favors speed of collection over database input so if you are trying to create operational procedures around data collection, the forced the consistency of traditional data storage can support that operational initiative in a way that unstructured storage may not.

Centralized vs. Federated data services

Another importance choice, especially for those that have satellite sites or more than one base of operation is the decision to create a single data service or to employ federated data services.

A *Centralized data service* is exactly what it sounds like, you are storing all of your data in one location. Single data services typically have good performance characteristics and since all the data is in your own database, you have more control over how to acquire and shape the data. However, this means that you also have all the responsibility for performing those manipulations locally. This is beneficial if your data partners consuming the information must depend on you for packaging that data payload in a formal structured format.

Federated data services gather analytical data from multiple related sources using an agreed set of rules. Typically, the consuming system doesn't store the data provided, such as in cases where blind data or aggregated data is all that can be provided rather than the raw data or identified data. It can also be useful in the case where you have multiple locations with separate data storage or acquisition methods. Federated data services also distribute the load of acquiring and shaping data between partners, so that no one partner has the entire performance burden of manipulating the data. Lastly, because you don't store all the data onsite, federated data services can make scaling data easier, you simply add more partners to the federation, or access more data from individual partners, instead of needing more space to store data.

4 DATA SHAPING

How do you best transform your data for analysis?

Data shaping is the process by which relevant data to your analysis question is pulled from the database and ordered in a way that you can use to answer your question in a reasonable timeframe. Data shaping involves further categorizing and aligning your data for analysis. For example, if you were looking at patient data, you might have test results in raw form, diagnosis and visit summaries from providers that include test that they ordered, the prescriptions that were issued, and the prescriptions that were filled. Now, let's say you wanted to know what patients were diagnosed with diabetes, prescribed a medication, and filled that medication. You might approach this by linking this information using one of the techniques described above and then transforming the information into a normalized collection based around the relationships between patient ids, diagnosis codes, test result codes, and visit dates

Both methods have different advantages and tradeoffs that must be considered.

HOW ARE YOU SHAPING THE DATA?

When employing a traditional data warehouse approach, data shaping is generally accomplished via extract-transform-load (ETL) operations prior to storage in a data warehouse. As mentioned above, data warehouses necessitate translating, reshaping, and manipulating data into a fixed and set structure defined by the storage schema. However, systems based upon NoSQL storage types generally defer the data shaping operations to the point in the process in which the analysis is occurring. Both methods have different advantages and tradeoffs that must be considered.

Data Warehouse vs. Unstructured Data Aggregation with Dynamic Data Shaping

Shaping data after collection allows for more flexibility by allowing you to easily store data that is related but not all in an identical format. You can use the same data (or subsets of the data) for different use cases, shaping only what you need on the fly, and not continually reorganizing how the data is stored to support your analysis needs. Instead, temporary in-memory data sets are built quickly, used to do the analysis, and then disposed. Unstructured storage techniques simplify storage, allow the system to take full advantage of all the ways in which the data can be queried, and allow for incorporation of additional disparate data sources into the analysis process without redesigning and loading a data warehouse. However, the architecture needed to efficiently shape unstructured aggregated data is very specialized and can be quite expensive to build and maintain. Thus the traditional methods of data shaping during the ETL are not without its perks. If your data sources are small and change infrequently data warehouse solutions are reasonably cost effective to implement and comparatively simple to maintain. Since all the data is already in the right format, ad hoc queries against it typically take less time to process. Additionally, since less shaping is done before passing data to the consumer, processing time can be streamlined there as well. With data warehousing you need to set up schemas for new data types and reconfigure your database before you gain access to new types of data.

If you are doing dynamic data shaping, can you do batch processing or do you require streaming analytics?

Typically, it is acceptable for data shaping to be performed with a batch approach on a set frequency. A batch job is executed with rules for shaping the data and the entire dataset is rebuilt during the job. The result of this data shaping is then typically cached in memory to be used by data analysis and visualization tools. Because of the performance needs, this process typically run on a periodic schedule while few users are on the. For operational data, a nightly update is generally more than adequate to meet the needs of a company. However, if you need your data updated more frequently at times when users are viewing the data a streaming analytics may be indicated. With streaming analytics, you reprocess the data on the fly depending on how the user requests it. Because of this, queries have to be focused so that processing is as efficient as possible. Streaming data can be performance intensive, but allows users to request up to the minute data from the system. It is important to know the answer to this question as it can greatly influence the selection of data shaping tools and technologies.

What tools appropriate for your data shaping choices?

Data shaping can be very complex, but as with other areas of software, there are a lot of tools that can help. Choosing the right tool or tools depends on a number of factors, such as the storage method or platform you chose and what kinds of data shaping and analysis you want to do. If you are doing traditional data warehousing, the data platform you choose typically dictates the ETL tools you can use. For example, Microsoft SQL Server has its own Integration Services platform (SSIS) that allow you to perform common ETL tasks in a tool that is very user friendly and requires just a base level of SQL knowledge for most tasks. If you are using unstructured data storage and taking the dynamic data shaping approach, there are numerous tools available that focus on solving different aspects of ad-hoc data organization shaping. Some examples that grew mostly out of the Hadoop infrastructure and all have slightly different strengths are Apache Storm, Spark, Pig, and Hive. These tools have been further extended and adapted to work with many of the NoSQL databases that exist. To further complicate matters some of these tools support multiple programming languages so you need to choose the best fit based on the capabilities of your development team and which of them best supports your analytics requirements. For example, Apache Spark supports Java, Scala, Python and R programming languages and use of each has their own use of pros and cons that needs to be considered. Other cloud platforms such as Microsoft Azure and Amazon Web Services have analytics tools that are optimized for working with their data platforms and are good options if you are exclusively using those platforms and need a high level of scalability. Typically, software development firms will have a preferred tool for each application and can help you decide which tool is right for you.

5 DATA SECURITY AND DATA GOVERNANCE

How do you make sure your data stays in your system and your users have appropriate access to the data they need?

Everyone wants to make sure that their data and their users are secure. When it comes to security, there's one golden rule: Don't write your own. In almost every case, it is better to use what is out there. The methods available on the market have already been stress tested by the public, thus working out most of the kinks and weaknesses. If you create your own, you run the risk of it failing at the first attack. That being said, the level of security you choose and the methods you employ will depend on the data you are securing. There are many different levels of security to choose from, but you'll want to weigh them against the ease of access for you users and the cost to maintain them going forward.

There are many different levels of security to choose from, here are some things to consider.

Encrypt your data –

When using a web API to transport data, the best security is to use TLS encrypted connection. TLS usually involves sharing public keys between sites. Once data is ready for transfer, a handshake takes place and private key is agreed upon. Then any communication that occurs uses the key decided on during the handshake. This process secures your connection between sites and prevents eaves dropping during transfer.

Only store the data you need –

In some cases, you need individual data points, but in the case where aggregated data will do, you can secure you data by only storing aggregated or blinded data on your site. While this doesn't secure the stored data per se, it does ensure that any information stored on your site can't be easily traced back to the original data, which by nature is more sensitive than the aggregate.

Use second factor authentication –

For some data, especially those that are subject to additional regulations such as HIPPA or PCI, additional security steps can be added that send information to phone or other device. The owner of that device then inputs the code back into the software. More secure second factor alternatives are apps like Google authenticator, which you download to a device where it generates a new 6-digit number every 6 seconds. The user then simply uses the generated code as authentication on the linked site. Lastly, there are traditional hardware methods of second factor authentication such as RSA key or cards that generate random numbers which can be used as keys for linked software or hardware devices.

Create different security levels for your users and use context –

By passing in a user's name and level of security, you can create context based views of data. In this way you can only allow certain people to see certain levels of data.

Don't leave breadcrumbs –

Make sure your user's can't tell what level they are or see anything in the web browser that would let them easily. As an easy example, thing of a web browser that contains the text "userid=1234". As a smart user, I might realize that changing my id to "1235" might change my security. As a rule, you don't want anyone to have easy access to ways to bypass your security measures.

Use alternate methods of securing an API –

API key: An API key is a randomly generated, user specific, string of characters and numbers. Each user on a site will have a unique API key which can serve to identify them to the site. When a user makes a call or request to the API, the key is sent along with the request and the API will use that key and give the user appropriate access. This type of security is typically used to give users access to related applications. This method streamlines access for users, but it is not the most secure because the key can be used by anyone that has it, not just the user it specifics.

Claim space authentication: Claims are basically a piece of information that tells something about a user. A claim could be the first name, a last name, or an id. Each application will have its own unique set of claims for users, and based on the information the claims contain, it will give appropriate access to the user. Claims are like defining a user role, but include all information about a user and are therefore more secure. Generally, each claim also comes with a token, or signature, of the authentication site. Claims can then be set to expire and then regenerated so that the token is different every time. Changing any part of the claim space, including the signature, invalidates the call, making this method more secure than an API key.

Not sure where to start? Still have questions?
Yahara has been helping clients with their data needs for over 20 years, so don't hesitate to reach out!
yaharasoftware.com

Want to read more from the experts at Yahara?
yaharasoftware.com/news

Yahara SOFTWARE

Words: Abbey Vangeloff
Images: Sandi Schwert
Collaborators: Adam Steinert, Patrick Cullen, Chad McKee, Kevin Meech, and Garrett Peterson
Sources: http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf