Dimensions for Characterizing Analytics Data-Processing Solutions

Jari Koister
August 2015

Introduction

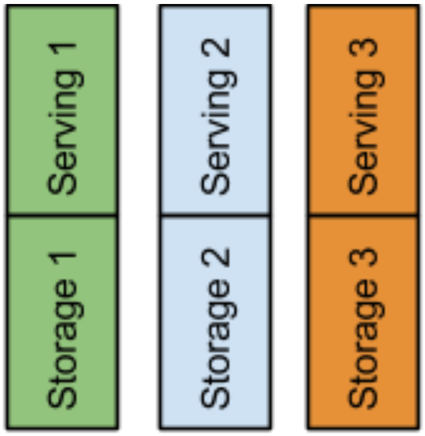
Data analytics and business intelligence have been of interest for a long time, and their needs have been addressed by various technologies based on relational technologies and cubes. Columnar databases and in-memory computing were developed to deal with increasing demand for quick response and interactive analytics. In recent years Hadoop was introduced to deal with very large datasets, which were previously difficult to handle with scale-up solutions. Even more recently, streaming and in-memory processing solutions have been introduced on inherently scale-out architectures. These solutions have been developed to deal with both speed and scale.

As a software architect and data analyst, it is notoriously difficult to determine what solution makes the most sense for a particular data and computation problem. It is made even more difficult because problems often evolve as we start working on them.

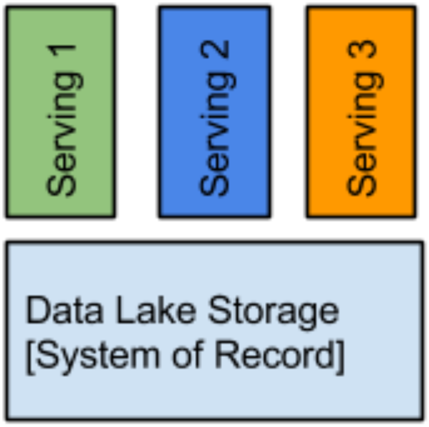
As an example, we often initially want some ad hoc analytics to explore and understand the data. Once we determine some interesting models, we often want these to be executable on a regular basis on some data arriving in the system. These two use cases are quite different. Another example is that we sometimes have a need for slow-moving data, while in other instances, new data arrive at high velocity.

There is clearly no one technology that provides an optimal solution for all problems. Traditionally organizations may have multiple technologies and systems dealing with different business problems. They may have a cube solution for business analytics while having a columnar-store solution for ad hoc queries on a subset of data. I call this the *parallel stack strategy*. It allows organization to meet different needs by providing different solutions, each with its own special advantages and disadvantages. The downside is that each has its own system of records; hence, there is no organization-wide record system. This usually also means that no solution can host all or even a large part of all the data used within the organization.

There are multiple emerging architectures for big data analytics. A unifying pattern is that that they have a data store for system of record stores and multiple stores that provide indexing and serving layers for different use cases. The idea is that the system of record is based on one of the modern scale-out architectures (typically HDFS) that allows an organization to host all data in one place in an economical way. Each serving component hosts a subset of the data or processed results that are solving a specific use case, again with its specific advantages and disadvantages. I call this strategy the *lambda strategy* after the reasonably well-known lambda architecture. I depict the principles below.



Parallel stack strategy



Lambda strategy

In the parallel stack strategy, as well as the lambda strategy, one needs to understand and pick the correct storage and serving technologies. In this paper we will discuss a framework for understanding the data at hand and the different technologies available, as well as the trade-offs involved in selecting one of them. Our goal is to characterize the problem's sweet spot for different solutions and to identify problems that specific solutions do not deal with particularly well. We do not expect the framework to be 100% complete, but we believe it will help in understanding what aspects are important in comparing and selecting technologies for analytics processing.

Data

It all starts with the data. A lot has been said about various ways of characterizing data, so we are not necessarily breaking new ground here.

The first basic characterization is the *structure of the data*. It can be derived from a relational model with a well-defined number of attributes for each record. Each attribute has a carefully selected type such as DATE, NUMBER, or URL. This is called highly structured data.

Unstructured data is commonly text for which we may need advanced techniques such as NLP to detect structure and meaning. Sometimes these texts are not even a proper language but rather slang expressions, such as tweets.

It is, of course, a continuum. I like to use the term *semistructured* for data that has somewhat of a structure, such as using JSON definitions. In this case there is an expected ordering or nesting of data, but many elements may be optional, making it tricky to parse. And the data in each attribute may not be explicitly type-checked when inserted, making it impossible to rely on properly formed data during the processing.

Another type of *mixed model* is when you have structured data but one of the attributes contains unstructured data such as text.

Size is a central property. Not all solutions can deal with very large datasets. But not every problem involves very large sets of data. So understanding the size of the data that you are dealing with is important. There is no reason to limit yourself to certain solutions if you do not need to.

Another important aspect of data concerns its *velocity*. We consider velocity the rate at which data are added to a dataset or data are changed. An example of adding may be making new purchases online. An example of changing data may be second granularity updates to a sensor in an airplane.

Some data are updated on a weekly or monthly basis. Other data are changing on an hourly or second-by-second basis. In the most extreme cases data are updated multiple times a second.

We will distinguish between *data sink latency* and *data source latency*. Data sink latency concerns the velocity of data as they come into your analytics systems. Data source latency concerns the freshness of the data generated from any analytics processes you have in place. Note that this is different from query latency, which concerns the time to answer a specific question. As the data used to answer queries cannot be more fresh than the data they are based on, the data source latency cannot be more recent than data sink latency.

The third aspect of data is data *quality*. This is to some extent related to the structure, but we consider it separately. Let's say you have a street address. It is very difficult to have a data type called "street address." This would require us to actually check that any data inserted into the attribute is indeed a valid street address, which is impossible for all practical purposes. So the street address attribute tends to be a string. There is implicit semantics indicated by the attribute name that cannot be formally checked by the system. Therefore, it expected that you see examples such as:

- Harrison St.
- Harrison Street.
- Harrisson St.

To clean up this data, one uses different algorithms to consolidate data fields so that more proper analytics can be performed. Another problem is missing values and how to deal with them. The quality dimension considers what quality of data we expect our computation to be able to deal with.

Finally, we consider *completeness* of data. The question here is whether the data are in a state that will enable you to do the analysis you are expecting to do. You may need to join different datasets to get where you need to be. Some technologies are better at combining datasets than others.

In summary, we characterize the data in terms of

- Structure: Structure, semi-structured, mixed-structure, unstructured.
- Velocity: High and low for data source latency and data sink latency.
- Quality: High quality means type-checked at insert, no missing values, and consistent formatting and semantics. Low quality means not type-checked, missing values, and inconsistent usage.

- Completeness of data. To what extent do you expect to need to join or enrich your data for analysis.

Computation

We use the term *computation* to refer to characteristics that we believe are essential in characterizing data analytics processing systems.

We have identified the following dimensions:
Selectivity: To what degree does the process filter data for processing? If a process selects and processes less than 20% of the data, we consider that a high level of selectivity. If the process uses 20–80% of the data, we consider that medium selectivity. If a process uses all or almost all of the data, we consider that low selectivity.

Processing Time: What is the desired processing time of a computation? If it can take hours, we consider that our system can allow for long processing time. If we expect results in minutes or tens of minutes, we consider that medium-length processing time, and processing that lasts 30 seconds or speed of thought is considered short processing time.

Query Execution Time: This dimension concerns interactive queries and systems from which we retrieve results rather than mainly computing new aggregations, joins, and so on. Obviously, interactive queries may involve such computations, but the main point here is that we at the time when we execute the query we may not know what execution time to expect. Interactive query execution concerns computations that we were not able to predict. A query with short execution lasts a few seconds or less. A medium-length query lasts up to a minute, and a long query lasts more than a minute.

Precision: Computation always comes with trade-offs. One such trade-off is precision. Some systems are designed to compute quickly, but they may end up with a result that is not completely accurate. In some cases this is a deliberate design goal because the system deals with samples or approximations to respond more quickly. In other cases it has to do with infrastructure limitations that allow for quicker processing but may, for example, result in some data being lost or double counted. We use three distinct values to categorize this dimension: exact, approximate, and lossy.

The final two dimensions for computation reflect common types of computations and whether they can be done efficiently.

Joins: Joins of datasets are common operations. In the context of large datasets, joins are particularly tricky. Skew in data and other factors can greatly impact the join performance. With some computational frameworks, joins in the traditional sense may be tricky or inefficient to execute. We consider support for joins advanced, basic, and none.

Aggregations: A common calculation is to aggregate data in different ways. Common straightforward aggregations concern time-series analysis. More complex computations create drill-downs and roll-ups that can later be navigated and even incrementally added to. Different solutions have different capabilities with respect to how well they support aggregations. It is hard to specify a scale for this capability. At a higher level we categorize as follows. A system supporting *advanced* aggregations provides ways of doing hierarchical roll-ups and drill-downs, enabling efficient ways of querying. A system with *basic* aggregation provides the means for counters and aggregations on single attributes. *Medium* capability is a broad range between advanced and basic capabilities and includes things like time-series aggregation.

The following table summarizes the dimensions, what they characterize, and the types of values we consider along the dimensions. Observe that this is not an exact science; rather, we use this to characterize and enhance our understanding of data-processing solutions for analytics.

Summary of Dimensions

Each dimension can be viewed from two sides. The first is to understand how your expected data and processing needs can be characterized with respect to the dimension. The second is how a specific technology meets your specific needs.

Aspect	**Dimension**	**Description**	**Values**
Data	**Structure**	What variety of data structure can the solution deal with efficiently and accurately?	Structured, semistructured, mixed-structure, unstructured
	Size	How big are your expected data, and how does that match what the technology can efficiently deal with?	S: megabyte M: gigabytes L: terabytes XL: 100s of terabytes. XXL: petabytes.
	Sink Latency	Velocity of data as it arrives in the system: How efficiently can the system ingest the data as they arrive?	**Very high**: greater than hundreds or more updates per seconds **High**: hundreds or more updates per hour **Medium**: data updated a few times on an hourly basis **Low**: data updated daily or less frequently
	Source Latency	The speed at which computations are reflected in the data that are used to power queries and analytics.	**High**: Analytics data are updated in real time as new data come in. **Medium**: Analytics data are updated on an hourly basis regardless of sink data latency. **Low**: Analytics data are updated daily or less frequently.
	Quality	Ability to deal with varying quality of data. Are the data conformant with the intended semantics of an attribute? We also consider missing values as part of quality. What is the share of values that are missing? Is that	**High**: Can handle and compensate for bad and low-quality data. **Medium**: Can handle bad data, but results

		significant, and how will it impact our analysis?	may not be reliable. **Low**: Requires data to be of high quality for results.
	Completeness requirement	Are the data self-sufficient in terms of what we expect to achieve? Or do we expect to join and enrich the data so that we can create the computations we are looking for?	Complete, incomplete
Processing	**Query Selectivity**	Does the solution deal with highly selective computations, or is it more suitable for low-selectivity computation?	**High**: less than 20% of data selected **Medium**: 20–80% of data selected **Low**: almost all data selected
	Query Execution Time	How quickly can the system provide a response to an interactive query?	**Short:** milliseconds or seconds **Medium:** speed of thought or at most 30 seconds **Long:** minutes or tens of minutes
	Aggregation	What is the complexity of calculations that can be performed efficiently and with reasonable implementation cost?	**Advanced:** advanced roll-ups with drill-downs, lattices and cuboids. **Medium:** over multiple dimensions **Basic:** simple counters
	Processing Time	For automated processing, what is the expected processing time?	**Short:** less than one hour **Medium:** less than 12 hours **Long:** > 24 hours
	Join	How elaborate is the support for executing a join over a variety of datasets with different size and distributions?	Advanced, basic, none
	Precision	What is the expected precision for computations? Will they always be exact, is approximation involved, or may some data be counted multiple times or lost?	**Exact:** always exact, includes full dataset **Approximate:** approximates through sampling or other techniques **Lossy:** may miss data due to loss or may count some data twice

Example Categorization

Below we identify some types of data-processing solutions and characterize them according to what they are best suited to deal with. Of course, people will claim that their solution can deal with all data and all kinds of processing. The point is to find the right solution for the problem at hand and understand what the trade-offs are.

Aspect	Dimension	RDBMS	M/R	Storm	Blink DB
Data	Structure	structured	all	all	structured
	Size	S->L	XL,XXL	S->XXL, streaming	XL,XXL
	Sink Latency	high	very high, high	very high	N/A
	Source Latency	high	medium, low	high	N/A
	Quality	medium	medium	high, low	low
	Completeness	incomplete	Incomplete	complete	complete
Processing	Selectivity	high, low	low	high, low	high, low
	Query Execution Time	short, long	long	N/A	medium
	Aggregation	advanced	medium	medium	medium
	Processing Time	short, long	long, short	short	short, medium
	Join	advanced	basic	basic	basic
	Precision	exact	exact	lossy	approximate