The following are comments by Frank Farance of the INCITS Ad hoc on Big Data. We did discuss this somewhat on our 5/6 weekly call, but I thought I’d upload for those who weren’t on the call.

Nancy Grady

I have a good number of problems with the Definitions document.  In short, the definitions and understandings are problematic and counter to the relevant and applicable sciences (terminology science, computer science, etc.).  Furthermore, this notion that all descriptive characteristics need to begin with the letter V (as in Victor) is terribly English-centric, which impedes international adoption, and artificially skews the terminology in to Cutesy Words, not words that have good, consistent, scientific meanings and basis.

Given that volume, velocity, and variety are in common usage, it would not make sense to use any other terms to represent those dataset characteristics. These words are of course English-centric, and other languages could use the appropriate translation of the terms into their language, or use the ones that are in common parlance already.

So in addition to the problematic terminology (concepts/definitions), the concept system is problematic, too.  In other words, it's not about getting to better definitions of "velocity" (etc.), the whole thinking along these lines is misguided.  As an illustration of what's wrong, look at the notion of "volume" and how it is a defining characteristic of Big Data.

On page 6, it says:

Definitions> "A difficult question is what makes Big Data big, or how large does a dataset have to be for it to be called Big Data? The answer is an unsatisfying "it depends." Data is considered big if the use of the new scalable architectures provides a cost or performance efficiency over the relational data model. In other words, the functionality cannot be achieved in a traditional, single platform relational database."

Well, duh.  That paragraph should tell one (as least in a terminological sense) that one doesn't have the right understanding if the result is "it depends".  Why?  Because terminology is about concepts, concepts have an extension (the objects that correspond to the concept), and the extension needs to be determined reliably, i.e., when two people conjure the concept, they are thinking about the same objects that correspond to that concept.

The question of whether data is “big” in the volume sense is a metric, and the question is whether your volume metric can be compared against a threshold to decide the data is “Big”. We have had ever-large datasets for years, so the question is when you need to use horizontally distributed resources to handle the data. Some large datasets can still be handled in traditional relational databases. Some not so large datasets are placed into distributed systems because of performance or cost. The question of whether you should use the new distributed technology is often a question of cost and performance, not that the data is so large it couldn’t be handled one way or the other.

The paradigm shift is to use the horizontally distributed resources for scaling, this is the concept.

The definition of Big Data refers to ones where your data characteristics push you into the distributed paradigm for cost/performance reasons.

As your code example below illustrates, a large dataset can be generated on one machine for a very long time, or distributed across many machines and be generated quicker. Which you choose is an economic or performance question. Perhaps this example is not purely a “big data” example, since it doesn’t require distribution across nodes to run in a reasonable length of time. You can certainly consider using big data technologies for “small” data to achieve faster performance.

An equivalent question would perhaps be ‘when does a large scale simulation need to be run in parallel across a number of nodes?’ If there is a metric (or metrics) over some characteristic that allows someone to decide if this simulation should run on an MPP machine or just a large server, then we can possibly figure out how the answer can be translated to data-intensive characteristics/metrics. I suspect the answer is the same, it depends. It depends on whether you can wait long enough, afford an even bigger server, or parallelize the code to take advantage of the many CPUs. I hope someone has some insight here because it would help us push our discussion forward on doing a better job of defining when we’re in the big data regime.

The other concern we’ve had is that we didn’t want to define big data in terms of the architecture used to handle it. We have that issue in using NoSQL (SQL the relational model query language) to describe a storage paradigm that is non-relational. We mention it, then suggest using non-relational as a better term. I have no delusions that ‘non-relational’ will ever replace NoSQL in public parlance. It’s become a term that has moved away from its original usage to by default mean non-relational storage models. We can suggest, however.

As an example of a poor concept, see the prior definition of "planet" which had a terminological problem because there was disagreement/inconsistency on the definition (i.e., it had a problem with the concept's extension).  The solution was not to define "planet" better, but to recognize that the problem was with the conceptualization of its concept system: instead of the concept system only having a notion of "planet" (whatever its definition), the better concept system is one that has \*two\* notions of "planet" and "dwarf planet".  With that different conceptualization, the two notions (two concepts) can be better defined -- the definition is better because of the concepts characteristics (the concept's intension) are more precise and the use of the concept to classify objects in the world (the concept's extension) is more precise/consistent.

Perhaps looking into the reason Pluto was removed as a planet could give us some insight as to ways to better answer the size question on whether data is ‘big’. Can we think of other terms to better categorize what technology solutions are appropriate. We haven’t gotten anywhere near far enough in discussing thinks like metrics or SLAs to address the ‘when is it big data’ question, but it would be great to make more progress on this fundamental question.

This is all basic Terminology 101 stuff, which should be applied.

Please pardon my next analogy because it refers to something horrific, but happens to be a good illustration of a terminological problem similar to the one we have in Big Data.  The crime of Rape is been defined in many legislatures as an Act Of Violence.  It turns out that this concept is problematic because there are some acts that are violent, but not considered Rape; and there are acts that are considered Rape, but not covered under this law.  From a terminological perspective, the "users" (society) have determined there is a problem with the extension of the concept (Rape) because there are objects inside the extension that don't belong (acts classified as Rape that are not considered Rape) and there are objects outside the extension that DO belong (acts not classified as Rape that are considered Rape).  Legislatures refer to "re-conceptualizing rape", changing it to an act against Sexual Autonomy, i.e., with better concept, the extension (illegal acts) can be determine!

d more correctly and more consistently.

My point here is: this kind of terminological conceptualization problem is just information science (e.g., other sciences, such as astronomy, have similar problems), and it isn't just the sciences (e.g., legislatures struggle, too).

So when we look at the notion of "volume" (one of those V-words), this is an example of an idea based upon incorrect conceptualization.  Here is an example (written in C) of something that would be considered "big data" because of "volume":

// adding two vectors, each a petabyte in size:

long long i;

for ( i = 0 ; i < 1000\*1000\*1000\*1000\*1000 ; i++ )

{

         A[i] = B[i]+C[i];

}

Yet, in fact, this is not a big data problem and can be performed on a single machine in about 10 days of computation.  Now maybe this kind of computation becomes a Big Data problem on some implementations, but not inherently because of volume, it because of something else.

So by the consideration of the multiple concept idea, this is large data, but could be handled by a single (vertical) system, or a distributed system. The data could be handled/generated either way, the question of whether you treat this situation as “Big Data” is a performance/economic one since it can theoretically be treated either way. Do we need some sort of “medium data” concept for this scenario?

So "volume" isn't an \*\*\*essential characteristic\*\*\* of "big data" (note: terminologists focus upon essential characteristics for good definitions).  And ditto for the other V's.

Correct, they are contributory, but the determination of the technologies to use is for performance not necessarily for the dataset characteristic of volume. In some cases the data can’t be handled on existing single systems, so the characteristics clearly push into the distributed regime. In other cases you may choose to distribute even if you didn’t have to.

So there are two possibilities: "big data" is a special kind of data (a specialization of the concept of "data"), or "big data" is a term that denotes something else that is not a specialization of the notion of data, e.g., possibly a field of study known as "big data".  I believe it is the latter: a field of study.

Big Data Engineering refers to the new technologies for distributing data-intensive systems, just as simulations went to MPP long ago.

In a lay sense, I believe Big Data is largely about thinking outside the relational model of databases.  If you Came Of Age (at least in information technology) after the early 1990's, largely you would have been exposed only to relational model thinking of data.  If you Came Of Age prior to then, your thinking of databases is likely to be broader, e.g., network model vs. relational model of databases.  And if you're a computer scientist who works outside of databases (e.g., operating systems, compilers/interpreters, and other systems work), then you have an even broader thinking about data structures and data processing (framed, typically, by Knuth volumes 1-3).

This is what we focused on, that Big Data Engineering is all about non-relational models.

Last year in the SC32 ad hoc on this work, I proposed the following definition of Big Data, which focuses upon four main characteristics: irregularity, parallelism, real-time metadata, and presentation/visualization.  Here is my suggestion:

Big Data: field of study based on convergence of problems in: (1) irregular or heterogeneous data structures, their navigation, query, and datatyping; [variety] (2) computation parallelism and its management during deployment or execution; [the distribution across nodes paradigm – the big data engineering] (3) descriptive data and self-inquiry about objects for real-time decision-making; [velocity for the latter real-time, but we don’t have a concept for this for the first part which is metadata] and/or (4) presentation and aggregation of data that exceed visual limitations of a single page [I would move this into a visualization skills issue, which was not in our scope. There are some visualization modalities that are better than others in rendering data representations to a screen, but this is outside our scope.]

We did not explore contributory metadata issues, nor have we fully explored the end-to-end time performance issues. Real-time analytics are quite different from batch historical analytics. This is on our list for our pattern discussions.

The goal in our NIST group was to make a definition refer to one concept, not several. You’re describing the characteristics of the big data engineering technologies, and these are precisely the types of pattern-determining characteristics that we wanted to address next…what are the implications to the usual concepts of ACID, etc in the distributed node storage. How do specific implementations differ in these characteristics?

Note 1: Big Data is not necessarily about a large amount of data because many of the concerns can be demonstrated with small (less than gigabyte) data sets.  Big Data concerns typically arise in processing large amounts of data because the four main characteristics (irregularity, parallelism, real-time metadata, and/or presentation/visualization) are unavoidable in such large data sets. This is why we defined big data as when you couldn’t avoid distributing across nodes.

Note 2: Computation parallelism issues concern the unit of processing (thread, statement, block, process, node, etc.), contention methods for shared access, and begin-suspend-resume-completion-termination processing.

Important characteristics to contrast between implementations; again a good discussion we need to have for patterns.

Note 3: Descriptive data is also known as metadata.  Self-inquiry is known as reflection is some programming paradigms.

We have had a lot of discussions about metadata, and we need more thought is to what aspects of metadata that have changed with big data, and which need to be discussed to complete the background needed to understand other big data concepts.

Note 4: The visual limitations concern how much information a human can usefully process on a single display screen or sheet of paper.  For example, the presentation of a connection graph of 500 nodes might require more than 20 rows and columns, along with the connections (relationships) among each of the pairs.  Typically, this is too much for a human to comprehend in a useful way.  Big Data presentation/visualization issues concern reformulating the information in a way that can be presented for convenient human consumption.

Exploratory Visualization has big data issues in the amount of data you’re trying to view/understand, and as you say in the ways to describe the dimensionality. Some things like your example above can benefit from a link-node kind of display with context-shifting to change the display to the node that is “central”. In terms of dimensionality, it takes pretty special displays to show the 4-6th dimension, more than that is extremely difficult.

I've pointed out the computational aspects of Big Data, including ...

Industry and standards work on parallelism, varying scope of computational units:

- thread parallelism: multiple threads running in the same process

- Industry and standards work on parallelism, varying scope of computational units:

- thread parallelism: multiple threads running in the same process

- block parallelism: multiple processors for an individual block { ... } of code

- statement parallelism: multiple processors lock-step executing statement-by-statement

- expression parallelism: multiple processors within a statement, e.g., matrix multiply X\*Y

- cluster parallelism: a process executing across a cluster of networked machines

We need to see what we can extract from the compute-intensive work on MPP systems and apply/adapt to data-intensive distributed systems to describe the patterns of parallelism.

These approaches have common concerns and remain today (as viewed by an implementer):

- How do I break up a task (algorithmically) into multiple processes? Or data across nodes?

- How do I map this efficiently to an execution/deployment architecture? MapReduce or other distribution mechanism?

- How do less expert programmers take full advantage of parallelism? More libraries than just MapReduce?

- What kinds of code (compiler) optimization are possible to match the hardware? How is this adapted for data query?

- What happens to the program/code if the hardware configuration changes, such as different number of processors and/or different connections among processors? Or if you loose a data node?

- How do I manage shared resources, e.g., locking/contention features? Or eventual consistency in a distributed system?

- How do the resulting parallel efforts merge back or aggregate into a single result? in MapReduce and/or other methods?

- How do I optimize the performance of aggregation?

- How do I manage Very Long Running Processes?

- How do I manage failure/reliability/redundancy?

These last are the same with data-intensive applications (or a cloud infrastructure) where you need to assure fault-tolerance.

In my paper, I also address several areas of improvement for descriptive data (metadata).  The following are non-exhaustive list of features to address Big Data needs:

We tried to be careful to restrict ourselves to what was new with the Big Data Paradigm, but our metadata description is way too light so far. We need to decide which of these we need to discuss in our definitions. In particular we explicitly deferred discussion of a data type taxonomy as being beyond the scope of our charter. The data types pre-existed the big data discussion, and I don’t think anyone came up with any new ones that have just emerged. Some data types are relevant to the discussion, as in XML or JSON for a data repository that stores the data in those formats, but for the most part, we didn’t include a data type taxonomy discussion.

Even with this comment, however, we need to revisit both metadata and data types to see if we can do a better job to what we have already.

- Support non-registry approach for metadata.  Right now, SC32/WG2 supports a metadata registry, but not metadata outside of a registry (ISO/IEC 11179, ISO/IEC 19763).  With the various datasets produced as intermediate results of Big Data computations — some only having a lifetime of milliseconds — one cannot expect every dataset's metadata to be registered in a centralized registry, i.e., metadata should be able to stand on its own, right next to the dataset it is describing.

- Support incremental/separable metadata.  Rather than a single comprehensive descriptive data (metadata) for an object, incremental pieces should be possible and an algebra of combinations of these pieces should be possible, which will afford further computability, interoperability, and re-use.

-Support graph, network, and other database models.  Many of the data topologies involve structures that do not neatly fit the relational model.  These kinds of data should have interfaces (creating/accessing/updating/deleting data, navigating data, querying data, etc.) that are harmonized with relational database interfaces.

We discussed these briefly as non-relational models, and wanted to dive into these in the second phase of the document.

-Support context descriptions.  Presently, the WG2 standard ISO/IEC 19773 supports a W5H  Context Data description.  Providing context is important for automated understanding and computation of datasets, especially datasets that are new and unfamiliar.   Other context descriptions are possible.

-Support standardization of context.  Just as Dublin Core provides a common set of elements and their metadata is further clarified by its vocabularies, context data needs further standardization efforts.  The UCORE efforts  provide one approach towards further refinement on context data, such as Where and When.

- Support common navigation of unstructured, semi-structured, and structured data.  If structured data is data whose navigation and datatyping (of the endpoint of navigation) are known, then semi-structured data is the condition where one but not both are known, e.g., a set of medical X-rays where the datatype of the X-ray is standardized, but the naming and nesting structure (navigation) of the X-ray data are different per-patient, and per-procedure.  Unstructured data involves data that may be navigated hierarchically, but whose datatypes and navigation paths are not known in advance.  Processing unstructured data is dependent upon common descriptive data (metadata) techniques and services (reflection).

-Support common reflection paradigm.  It should be possible to ask any datum (dataset) or object questions about its nature, such as: available attributes, available services/methods, type signatures for services/methods, etc..

- Support common parallelism/clustering, aggregation, locking primitives.  The discussion of parallelism above frames this topic.  These features should be available in small scope (e.g., threads of a process), and large scope (e.g., across servers of a wide-area network).

One that definitely needs discussion in implementation patterns.

- Support common parallelism/process/reliability management primitive.  Aside from the processing, the management of the processing needs a common solution.

-Support common datatype naming conventions.  Because agreed-upon datatypes  are fundamental to common processing and interoperability, these (and their spellings) need to be standardized across a community of practice.

Agreed, but to a large extent more than we can tackle since this is not new to the big data paradigm

-Support common attribute attachment/association conventions.  Metadata, attributes, and such need to be attached and associated via common methods, e.g., headers, URIs, links, and such.

An important area for standardization discussion