

Three Experts on Big Data Engineering

cover image here

We posed Clemens Szyperski (Microsoft), Martin Petittlerc (IBM), and Roger Barga (Amazon Web Services) these questions:

- What major challenges do you face when building scalable, big data systems?
- How do you address these challenges?
- Where should the research community focus its efforts to create tools and approaches for building highly reliable, scalable, big data systems?

We hope you enjoy their answers. —*Guest Editors*

Dealing with the V's of Big Data

Clemens Szyperski

“Big data” is a captivating term. People have used it to describe a range of phenomena, often characterizing it according to a number of v's, starting with the traditional velocity, volume, and variety. Other dimensions have been added, such as veracity (the data's degree of truthfulness or correctness). In essence, big data is characterized as a high bar on all these dimensions. Data


arrives at high rates, appears in huge quantities, fragments into ever more manifestations, and still must meet high quality expectations.

Engineering systems that meet such a broad spectrum of requirements aren't meaningful as such. Instead, you must narrow the focus and ask what the particular system to be built is supposed to address. For instance, the service I work on (Azure Stream Analytics, a platform service in the Azure cloud) focuses on velocity because it supports stream and complex event processing using temporal operators (up to 1 Gbyte/s per streaming job). Volume, in the form

of state and reference datasets held in memory, is significant too, but in ways quite different from mass storage or batch-processing systems. In the presence of latency expectations (end-to-end latencies in the low seconds) and internal restarts to meet fault tolerance requirements, veracity comes to the fore. For instance, today output meets an at-least-once bar, but exactly once would be nice and is challenging given the variety (oh-oh, another v!) of supported output targets. Talking about variety: Besides the richness in data sources and targets, the nature of very long-running stream-processing jobs also requires flexibility in dealing with evolving schema and a multitude of data formats.

It's fascinating to examine the technical challenges borne by relevant combinations of requirements in velocity, volume, veracity, variety, and other dimensions. However, to be more than fascinating, the combinations must address particular audiences' needs. Given the impossibility of meeting maximal requirements in all dimensions, big data, more than any other engineering category I've encountered, faces a deeply fragmented audience. From traditional hard-core distributed-systems developers, to data developers, to data architects, to data scientists, to analysts, to developers of larger end-to-end solutions in spaces such as the Internet of Things, the list is long.

Just as maxing out on all dimensions is impossible, it's impossible to meet all these audiences equally well with a single product or small set of products. For example, we've designed Azure Stream Analytics to be high-level, with a declarative language as its main interface, and to serve a broad set of customers who aren't distributed-systems



developers. A service that's high-level and composable with many other services (like any platform service must be) shouldn't expose artifacts of its internal fault-tolerance techniques. This leads to requirements of at-least-once (or, ideally, exactly once) delivery, repeatability, and determinism. These requirements aren't specific to big data but invariably become much harder to address when you're facing the dimensions of big data.

So, a large part of the engineering challenge, and one worth tackling in forward-looking research, is to construct larger big data solutions (such as services) out of composable elements to reduce the high cost of engineering those solutions. Starting with the fabric to manage resources, the trend is pointing toward cloud oceans of containers—moving from (virtual) machine to process-level abstractions. Even at this level, challenges abound if we want to map work run on behalf of multiple tenants onto a single such ocean. (Container oceans are the natural resources to drain your data lakes into!) On top of such infrastructure, we must address the core challenges of affinitizing computations to the dominant resource. That resource might be storage hierarchies or network capacity and might require either wide distribution for load balancing or collocation for access efficiency.

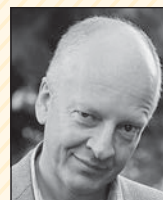
Given such a fabric, we then must systematically build highly specialized microservices that tie various “knots” by addressing specific sets of requirements. Just as with components, where we might have hoped for the definitive set of building blocks from which to compose all applications, we might hope for a closed or almost closed set of

microservices that will be the definitive platform for composing big data solutions. That's unlikely to happen—just as it didn't happen for components.

In this complex space, we need research into better ways to manage resources (oceans) to address contradictory requirements of collocation, consistency, and distribution. Abstractions of homogeneity break down as containers end up allocated on hardware hierarchies and software hierarchies with networking infrastructure that's far from ideal crossbar switches. If this weren't enough, the need to process work on behalf of possibly malicious or mutually hostile tenants requires deep security and privacy isolation while retaining flexible resource allocation and avoiding layers of internal resource fragmentation (a source of major resource inefficiency). Such fragmentation is traditionally the case when you rely on isolation at the virtual-machine-stack or hardware cluster levels.

Today, we're somewhat halfway through the research journey I just sketched, by building platform services that focus on individual sets of traits, that compose with each other, and that in combination can meet a variety of needs. However, these services are the product of several competing efforts, leading to overlapping capabilities, often limited composability, and confusion for those who wish to build solutions. Just in the realm of streaming technologies, we have not only various open source technologies, such as Apache Storm and Apache Spark Streaming, but also the various proprietary technologies found in the public-cloud offerings. Azure Stream Analytics is just one of the latter. This richness of choice will continue to be

with us for quite some time, leaving such systems' users with a dilemma of choice.



CLEMENS SZYPERSKI

is the group engineering manager for the Azure Stream Analytics platform service in the Microsoft cloud.

Contact him at clemens.szyperski@microsoft.com.

Changing How We Interact with Data

Martin Petitclerc

There are many technologies for big data engineering, and no one technology fits all needs. An important difference exists between tuning a system for a specific dataset (repeating the same jobs) and having a system that tunes itself on demand (ad hoc) on the basis of different datasets and different queries against them. As the volume, velocity, and variety of data grow, the goal is to not just handle more data but also find ways to reduce the human intervention necessary to get the desired information. Rule-based approaches—for example, ETL (extract, transform, and load)—aren't sustainable. We must change how we interact with the data.

As the amount of data grows, the amount of potential information grows. All potential pieces of information aren't equally important to everybody, and their value could change over time. Something unimportant today could become

important tomorrow, whereas other pieces of information (for example, security breaches) are always important. It's about getting the right piece of information at the right time.

Currently, we address those challenges by bundling different technologies for different needs—for example, traditional relational databases with emerging big data technologies. Still, these systems aren't getting simpler but are becoming more complex to develop, tune, and sustain, multiplying the technical challenges.

Involving cognitive systems in all phases of the data process is the way to reduce human intervention. It's also a way to link the data to users' tasks, objectives, and goals, defining all together the user's current interest in the data or the user's context for the system.

Systems that can understand those tasks, objectives, and goals and what's relevant over time will more effectively serve users' daily needs for data, information, and facts. Such systems won't overload users with irrelevant or unimportant things. For example, consider getting a summary every morning about all the changes you need to know regarding the current week's production objectives. This information includes root cause analysis and action recommendations on divergences, with impact analyses detailing how each of those actions would impact the outcome.

Such systems should empower everybody to understand data without them having to become a data scientist or an IT person. This includes simplifying complex tasks such as joining structured and unstructured sales data to compare customer sentiment with sales figures, including their correlation over time.

Another such task is semiautomated data cleansing that applies a set of relevant actions on the required data at the required time. This is probably better than having the IT folk prepare a large amount of data that might never be used because the users' needs change before the data is even ready. Also, data cleansing can't occur in a black-box manner, and data lineage is important so that the users can understand what was done, why, and how the transformation affected the data.

The idea is not to replace data scientists but to free them from supporting basic activities and let them focus on work having higher value to their organizations. For example, they could build a more accurate model to compute future insurance claims that incorporates climate change information. Everyone throughout the organization could then use this model to perform forecasts.

Privacy will be a challenge for such data analysis power, as the amount of available data grows. For example, attackers could still reconstruct information indirectly even if privacy was protected at diverse individual access points. They could link geospatial and temporal data to other data and correlate all the data to identify an entity (such as a person).

The research community should focus on simplifying the handling of data so that it's more contextual and on demand, without requiring IT intervention at all stages of the process. The community also must determine how cognitive systems can empower all types of users in an environment in which the volume, velocity, and variety of data are constantly growing. Important research areas include user interaction with data; data

lineage; automation; visualization; structured and unstructured data; data manipulation and transformation; educating users about findings; and the ability to extend, tune, and further extend such systems.

Today, the focus on big data seems to mostly involve performance, but empowering people to quickly obtain information is what will make organizations more effective.



MARTIN PETITCLERC is a senior software architect at IBM Canada for Watson Analytics.

Contact him at martin.petitclerc@ca.ibm.com.

Coping with the Scaling Cliff

Roger Barga

Big data and scalability are two of the hottest and most important topics in today's fast-growing data analytics market. Not only is the rate at which we accumulate data growing, so is the variety of sources. Sources now span the spectrum from ubiquitous mobile devices that create content such as blog posts, tweets, social-network interaction, and photos, to applications and servers that continuously log messages about what they're doing, to the emerging Internet of Things.

Big data systems must be able to scale rapidly and elastically, whenever and wherever needed, across

multiple datacenters if need be. But what do we really mean by scalability? A system is considered scalable if increasing the available resources results in increased performance proportional to the resources added. Increased performance generally means serving more units of work but can also mean handling larger units of work, such as when datasets grow.

You can scale up by adding more resources to existing servers or scale out by adding new independent computing resources to a system. But eventually you'll run out of bigger boxes to buy, and adding resources will fail to return improvements—you'll have run off the edge of the scaling cliff. Scaling cliffs are inevitable in big data systems.

A major challenge in achieving scalability and the opportunity to push scaling cliffs out as far as possible is efficient resource management. You can shard your data, leverage NoSQL databases, and use MapReduce for data processing until the cows come home, but good design is the only way to ensure efficient resource management. Efficient design can add more scalability to your system than adding hardware can. This isn't limited to any particular tier or component; you must consider resource management at each level, from load balancers, to the user interface layer, to the control plane, all the way to the back-end data store. The following are select design principles for resource management to achieve high scalability.

Asynchronous versus Synchronous

Time is the most valuable resource in a big data system, and each time slice a thread or process uses is a limited resource that another can't use. Performing operations asynchronously

will minimize the time a server is committed to processing a request. Servers can then queue long-running operations for completion later by a separate process or thread pool.

Sometimes, a system must perform operations synchronously, such as verifying that an operation was successful to ensure atomicity. Carefully differentiate between system calls that must be processed synchronously and calls that can be written to an intent log and processed asynchronously. This principle can also eliminate "hot spots" in a big data system because it enables idle servers to "steal" work from the intent log of a server under a high load.

Dealing with Contentious Resources

All systems possess finite physical resources; contention for these resources is the root cause of all scalability problems. System throttling due to insufficient memory, garbage collection, or insufficient file handles, processor cycles, or network bandwidth is the harbinger of an impending scaling cliff.

design can begin with quick SSL (Secure Sockets Layer) termination at the load balancer. Hardware load balancers have crypto cards that can terminate SSL efficiently in hardware and decrease the front-end server load by as much as 40 percent. The fast SSL termination will also increase client performance. You can apply this principle throughout the system layers.

Logical Partitioning

Logically partition resources and activities throughout the system, and minimize the relationships between them. Partitioning activities can help ease the load on high-cost resources. A best practice is to logically partition your application between the proxy or user interface layer, control plane layer, and data plane layer. Although logical separation doesn't mandate physical separation, it enables physical separation, and you can scale your system across machines. By minimizing the relationships between resources and between activities, you minimize the

Good design is the only way to ensure efficient resource management.

A design principle is to not use a contentious resource unless absolutely necessary, but if you must use it, acquire it as late as possible and release it as soon as possible. The less time a process uses a resource, the sooner that resource will be available to another process. Review code to guarantee that contentious resources are returned to the pool within a fixed time period. This

risk of bottlenecks resulting from one participant of a relationship taking longer than the other.

Partitioning also lets you identify metrics and measure utilization at each layer. A front-end proxy layer that handles incoming requests might best be optimized for transactions per second, and the control plane that manages operations might best be optimized for CPU

utilization, while the storage plane might best be optimized for I/O operations per second. This lets you ensure your system is balanced, with no single layer presenting a bottleneck or an overabundance of resources, the latter of which can result in underutilization or put pressure on other layers in the system.

State Caching

Employ a state-caching fleet. If at all possible, avoid maintaining state, which consumes valuable resources and complicates the ability to scale out. However, sometimes you must maintain state between calls or enforce service-level agreements. State shouldn't be held by a single resource because that increases the likelihood of resource contention.

So, a best practice is to replicate state across servers in the same logical layer. Should the server come under load and be a point of resource contention, other servers in the same logical layer can continue the session by using the state in their cache. However, peer-to-peer gossip protocols can break down at large scale, so a small ($\log N$) dedicated caching


fleet is required. Each server persists state to a single server in the caching fleet, which then disseminates this across a quorum in the fleet. These servers can lazily propagate state to servers in the logical layer in an efficient and scalable manner.

Divide and Conquer

At some point, all big data systems will encounter a scaling cliff that can't be engineered around. The only resort is the time-proven technique of divide and conquer: making a problem easier to solve by dividing it into smaller, more manageable steps. Just as your big data system is logically partitioned, possibly into microservices, you create a separate instance of your system to achieve massive scale.

Automatic Resiliency

There are many open challenges on the road to more advanced and scalable big data systems. One challenge that warrants further research is automatic resiliency. A well-designed big data system can be resilient enough to withstand the unexpected loss of one or more computing

resources. But a truly resilient system requires both good design and service-level support to automatically detect and replace instances that have failed or become unavailable. When a new instance comes online, it should understand its role in the system, configure itself, discover its dependencies, initiate state recovery, and begin handling requests automatically. 



ROGER

BARGA is general manager and director of development for Amazon Kinesis data-streaming

services at Amazon Web Services. Contact him at rsbarga@gmail.com.



Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.