The Problem with Hadoop

Hakan Jakobsson
hakan@hakan-jakobsson.com

Version 2015-08-24

Introduction

Hadoop has been one of the hottest buzzwords in high tech in recent years. One of the vendors in the space, Cloudera, has attracted over $1 billion in venture funding, but there are numerous other players on the Hadoop bandwagon. Hadoop is about Big Data and scalability, important concepts in a world of ever-increasing data volumes.

There have been some chinks in the armor, though. Recently, questions have been raised about whether Hadoop is really taking over the world. A Gartner survey found investments in Hadoop to be tentative.
http://www.gartner.com/newsroom/id/3051717

On a brighter note, when Hortonworks reported its 2015 Q2 numbers, revenue was $30.7 million, up 154 percent from the same quarter in 2014 and beating analysts' expectations. On the other hand, the operating loss on that revenue was $42.1 million compared to a $23.4 million loss a year earlier. Critics have also questioned whether Hortonworks open-source approach can be profitable. Its two major competitors, Cloudera and MapR, have substantial proprietary components in their portfolios of offerings.

So an obvious question is what will determine the ultimate success of Hadoop. What are the obstacles on the road to becoming a widely used mainstream technology that could limit it to a niche? First, one might want to ponder the question:
"What exactly is Hadoop?"

What is Hadoop?

Hadoop is an ecosystem of technologies that provide a framework for large-scale computing. There are so many of them that trying to generate a complete picture would be a challenge, but some components are more "core" than others. But the question of what Hadoop is may not have a completely trivial answer.
http://blog.cask.co/2015/07/what-is-hadoop-anyway/

Right there, we have stumbled on a problem: Fragmentation. Some of the many components of the Hadoop ecosystem have overlaps in functionality. Often, such overlaps are the result of the original technology having some deficiencies and of newer technologies that came about to address those problem. So now you have entrenched technologies battling it out with newer ones that may be better in some circumstances making technology choices within the Hadoop ecosystem less straightforward affairs.

But let's look at what originally made Hadoop so attractive. Among the very core components are HDFS, a distributed file system, and MapReduce, a programming model for distributed computing on large clusters. These are very key components if you want to run massive computational tasks on very large clusters built from inexpensive commodity hardware – the very key to the allure of Hadoop. You need a distributed file system and you need to be able to program your computational tasks to run on possibly many thousands of cluster nodes.

Let's start with HDFS. A distributed file system is a good thing to have. Wikipedia lists about a dozen just as examples. HDFS obviously gets the job done, but one of the major Hadoop distributers, MapR, was sufficiently unimpressed with HDFS to come up with its own proprietary implementation with the same external interfaces. But HDFS is there, it pretty much works, and it's very much a key component in the Hadoop ecosystem.

The limits of the computing paradigm

So here is where the real problem of Hadoop resides. It's not with the file system; it's with the computing paradigm(s). And there are a few that have emerged as alternatives to the original MapReduce framework in order to improve upon it in certain ways or complement it in areas where MapReduce is unsuitable. You have Spark, Tez, and Storm, to name a few, in addition to a variety of storage formats. All this is both good and bad in that it gives users more options for a suitable combination of technologies, but it also introduces confusion and fragmentation.

Hadoop, as a paradigm, assumes that you want massive scalability inexpensively and addresses that need by providing a framework for scalable computing running on massive clusters of commodity hardware. That's great in theory, but let's examine the limitations. What size is the market that really needs that much data and processing power? Do you really need something akin to Hadoop unless you are a Google, Yahoo, or Facebook? Sure, there are many other players with a lot of data, but how big is that market opportunity realistically in comparison to the overall IT market? For every Facebook, there are probably a fair number of midsize companies with more modest requirements.

Here is where things get problematic for the Hadoop paradigm: It is inefficient. It sacrifices efficiency for scalability. The basic concept is very simple. There are algorithms that are extremely stupid but scale very well. Essentially, they are very

dumb but due to their simplicity, they can easily be parallelized over a large number of cluster nodes. More clever algorithms may exist but will eventually run out of gas when being processed on massively parallel hardware architectures.

So where Hadoop shines is at scaling often primitive algorithms by brute force. The old phrase "throwing hardware at a problem" comes to mind, but at least it's relatively inexpensive hardware.

I went to some of the relatively early Hadoop World conferences in 2009 and 2010 sponsored by Cloudera and took some of the classes that were offered. One notable thing about the early classes (other than that Hadoop was somewhat lagging in security technology) was the preferred way of implementing things algorithmically. For the single-source shortest-path problem in graphs, a massively parallel breadth-first search was advocated. That's an extremely wasteful approach, but it's an approach that is scalable. A more clever approach would be to use Dijkstra's algorithm. However, part of the cleverness of that algorithm is that it uses a global state, something that is hard to do efficiently if you want to scale a cluster to an arbitrary large number of nodes in a distributed architecture.

The distinction between scalability and performance have been studied by people who have noted that, while a stupid algorithm may scale well, running it on hundreds of cores on a cluster may not give the same performance as a smarter algorithm running on a single core on your laptop.
http://blog.acolyer.org/2015/06/05/scalability-but-at-what-cost/
One would think that it defeats the purpose of running on scalable, inexpensive, commodity hardware if your computational framework is very inefficient.

So the next obvious question…

Is a very inefficient way of computing really a good idea? It probably could be if your life depended on delivering a result back really quickly no matter what the scale of your data set. But what if you are a CIO that has to consider the effectiveness of the dollars that you spend on infrastructure? Let's say your paradigm for computing is both highly scalable and highly inefficient at the same time, and the scalability is only needed for a small subset of tasks. Couldn't that lead to a very inefficient way to spend you IT dollars, especially if the computing paradigm is somewhat nonintuitive and with a limited skill pool?

So to what extent do you really need all that scalability? A Cloudera survey from 2012 studied the properties of the workloads of a few Cloudera customers.
http://blog.cloudera.com/blog/2012/09/what-do-real-life-hadoop-workloads-look-like/
It found that the participating customers moved 27 PB of data over 73,836 jobs. Data "moved" includes the total of input, output, and shuffle sizes. How much is that on average per job? It comes to 366 GB – less than the size of the SSD drive of the laptop that I bought the same month the survey was published. And it stands to

reason that even if a company has accumulated vast amounts of data, not all computational tasks will involve all of it. More likely, like in the Cloudera survey, the average tasks will only involve a fairly pedestrian amount.

So then, how do you justify using a computationally inefficient framework when your typical task doesn't require all the scalability that the framework is all about? This is a key question. You could try to justify it by claiming that all data should go into one gigantic cluster in its raw format, a "data lake" serving vast numbers of users and accessible to different kinds of processing within the Hadoop ecosystem.

The data lake

So what is wrong with the concept of a data lake other than that Gartner will tell CIOs that it's a bad idea?
http://www.gartner.com/newsroom/id/2809117

Putting a vast amount of raw data in a huge cluster doesn't add a whole lot of value by itself. What about security? Hadoop is hardly the poster child for great security and the idea of lumping all your corporate data into one big pool is something that should make any CIO think twice.

Security comes at many levels in an IT infrastructure. For instance, relational database products have pretty well established security models, but those policies work based on the data being interpreted as rows and columns of tables. That interpretation doesn't exist in the raw data – you need the database for that. An HR application may implement various security related features. What's the process for changing someone's salary? A manager can presumably see the salaries of his reports, but are their social security numbers visible to him? There is no logic for policies like that in the raw data, only in the HR application. So if we have a large Hadoop cluster with massive quantities of raw data, what part of it can be accessed by whom? As far as I know, there are currently no HR applications in the Hadoop ecosystem and why would you need a gigantic cluster for your HR processing?

Even worse, the idea of having multiple types of Hadoop processing access the same data doesn't go very well with security. What is the point of carefully defining what database columns a Hive user can access if the same user can see the entire underlying HDFS files when running MapReduce jobs? Security models that are based on a certain interpretation of the data, e.g., as rows and columns in a database table, aren't easily sharable with processing models that view the data according to some other paradigm. The data lake is supposed to support storing the data in its original format, which could be XML, JSON, CSV files, any number of proprietary formats, etc. Imagine all the patient data of a hospital being stored in its raw format in a data lake and available for any type of analysis. How do you protect patient privacy in an intelligent way? The risk is that the system has to be locked down in such a coarse and draconian way that meaningful analysis of the data becomes impossible and the data lake becomes a data graveyard. Still, that might be

preferable to an Ashley-Madison-caliber security breach, something that the data lake concept seems to be inviting.

There is some logic to how the data lake concept gained traction among Hadoop proponents in addition to being an excuse for the need of gigantic clusters. ETL was an early success story for Hadoop. ETL often has very strict SLAs in scenarios where data from operational systems has to be cleaned, transformed, and consolidated for analysis within some tight time windows. Hence, there is a real need to be able to provide a massive amount of computing power for what may only be a few hours each day. Traditional ELT solutions have been costly so it's easy to see the draw of a scalable, open-source framework running on inexpensive hardware. So if the use of the Hadoop cluster is restricted to very specific ETL processes where the security and consistency of the data, from an end-user perspective, is completely controlled by the database systems where the data is landed, Hadoop seems viable. The general idea of using a Hadoop cluster as a temporary repository for storing and processing operational data for restricted purposes and with restricted access makes sense.

However, using the Hadoop cluster as a more general backend for processing of all your corporate data is a different ballgame. But it's a natural direction if you want to justify the need for large Hadoop clusters. The problem is that any step towards centralizing data processing brings with it certain flavors of a Soviet-style planned economy with five-year plans. Ironically, you would generate the exact same issues that have lead people to bypass the bureaucracy of traditional IT to create "shadow IT" as a way of getting things done more swiftly. One would think that there is some kind of irony in that many early adopters of Hadoop probably would be in the camp that would eagerly bypass central IT only to find themselves involved with a technology where supporting centralization is a major priority.

YARN

So if you need to support a large number of computing tasks going against a large infrastructure for computation, you need to figure out how to assign the computing resources. Usually, the demand for computing resources will exceed the supply and any attempt to increase the supply will often result in creative users finding ways to generate additional demand as previously impractical tasks become more feasible. Hence, the need for pricing mechanisms such as chargebacks to balance supply and demand and make a particular organization pay for its share of some common resource it has been using.

There have been different ways of allocating computing resources. Often, decentralized approaches evolved as departments realized they had a specific need and got individual resources based on that need. In data warehousing, for instance, the data was typically based on operational systems that may or may not have been capable of doing analytical queries. But even if the operational systems had the capability of supporting analytical queries, the impact on the performance of the operational system could be disruptive. Better just offload the data to a separate

system optimized for analytical workloads and have the organizations using them pay for the hardware and software. And here, ETL became a crucial component.

In a more centralized approach, you run into problems that go way beyond who is going to pay for the storage and processing of the data. How are you going to allocate finite resources between tasks with different requirements when it comes to throughput, response time, latency, etc., tasks that may emanate from different organizations where tasks may have different priorities even when generated by the same user? In a decentralized approach, the individual organizations using the data would figure out how to deal with their processing needs provided that you could supply them with the underlying data. In a centralized environment, you need a sophisticated apparatus.

2015 marks the 20th anniversary of the first version of the functional specification of Oracle's Resource Management feature. It was meant to provide a mechanism for managing the resources of a relational database system, which is a significantly more restricted environment than a whole computational framework like Hadoop. While the first version of Oracle's feature was released a few years later, as far as I know, 20 years later, there is still a group at Oracle working hard to perfect it. YARN is the major Hadoop resource management project.

So what makes resource management such a complicated proposition? First of all, what resources are we supposed to manage? CPU? Memory? Disk I/O? Network? Disk space? Others? If we are going to have individual policies for different types of resources that may very well interact, we have a bit of complexity right there. You might start out using CPU as a convenient proxy for resources only to find that some customers want to be able to manage memory or disk I/O as well, and access to external networks comes with its own set of issues. Moreover, there are ways to globally optimize the distribution of memory between concurrent operations in order to minimize the risk that an operation, like a sort, may need to spill intermediate results to disk. A manually set memory management policy could well be highly suboptimal from the standpoint of global throughput.

Once you have figured out the resources you want to manage, you need to figure out the mechanism for how to allocate them. That involves figuring out both the physical implementation, like how to make the CPU devote work to some tasks in accordance to some policy, as well as the abstract mechanism for defining such policies. How do you schedule resources to avoid starvation issues? Do you schedule based on priority or emphasis, i.e., do you have a model where task A could have priority over task B for everything or one where task A could get 80 percent of the resources and task B be guaranteed 20 percent? How do you classify the importance of tasks? Is it by user, groups of users, or by type of task? If you support managing resources both by organization and type of task, how do you deal with organizational hierarchies and task hierarchies and how do you reconcile the two (which you must since you are going against the same set of resources)? Can the same user run different tasks with different levels of priority? If a certain group of

tasks are in the same importance category, do you schedule CPU resources between them round robin or do you prioritize tasks that have run for a long time so that they can complete? Is setting resource policies the prerogative of only a central authority or can it be delegated within some centrally decided set of limits? If so, what is the mechanism for delegating authority? How does a user request a high priority for an urgent task if lacking the authority to set it? If a task or group of tasks is guaranteed a certain amount of resources but doesn't use fully use them, how do you divide the leftover resources among other tasks? Do you address the requirements of an important task by letting it run with an average priority on a larger than average number of nodes or with a higher than average priority on an average number of nodes? How do you prevent the system from getting overloaded and thrashing, e.g., do you cut back on the amount of resources that each task can use or do you have a queuing mechanism where new tasks aren't let in until there is room for them? If you have a queuing mechanism, how do you define queuing policies? With queuing, you definitely need to address starvation issues to prevent lower priority tasks from getting stuck at the end of the line forever.

You can probably come up with reasonable use cases for pretty much any possible answer to the questions above, so a resource management system with full-blown functionality is likely to be somewhat complex to implement. And, of course, the resource management system needs to play along with your security mechanisms, and you need to have a user interface for the whole thing.

Unfortunately, implementing the resource management framework is just part of the equation. In order to use it effectively, users have to understand, not just the technical details of the feature, but also all the ramifications of turning a certain nob this way and another one that way. And then, if this resource management framework is supposed to be used for a massive cluster used by a large number of organizations generating different tasks of different levels of importance, the people running the whole thing have to be able to map a complex set of computational requirements onto a complex mechanism for managing resources in a centralized fashion (since the cluster is a centralized resource). Perhaps not the best recipe for agility.

The reality

The reality is that the vast, vast majority of valuable computing is done in software that was never developed to run on a Hadoop platform. SAP Manufacturing, Oracle Financials, Salesforce SFA, Microsoft Office – do they run on Hadoop? No, and why would they need to? The more likely scenarios for Hadoop are where the requirements go beyond the capabilities of traditional infrastructure or where the traditional infrastructure is simply overly expensive. The former case is a niche in comparison with all of IT spending and one where players will often create their own in-house solutions. The latter case is limited as well. Back during the height of the original dot-com boom, there were many startups that were running expensive proprietary solutions, like Oracle on Sun Solaris on expensive SMPs. Today, startups

might be running MySQL on Linux on inexpensive commodity hardware. The latter approach with a less expensive open-source stack may be just fine for a lot of scenarios that don't need unbounded scalability and the headaches that come with the type of programming paradigm needed to support it.

So the likely prospects for the success of Hadoop will be intimately tied to the growth of data volumes that would overwhelm traditional solutions. A bet on Hadoop is essentially a bet that social networks, location based data, the Internet of Things, and all the other usual "Big Data" suspects will deliver data volumes that go far beyond what more traditional techniques can handle and not just for some niche companies with extreme requirements. Moreover, it assumes that those data volumes can't be shrunk by inexpensive technologies that will either filter or compress the data at the edges reducing the need for all of the large volumes of generated data going into some central repository. For an example of the last point, let's say that I wear a device that is capable of detecting and passing on my exact GPS coordinates every 10 seconds. If I just sit on a park bench reading a book for two hours without moving a centimeter, would it make sense to store all those raw GPS readings in their full glory in a gigantic data lake? Most likely, other inexpensive technologies would have shrunk the amount of data that needs to be propagated to a central environment. When calculating the impact of the possibility of large amount of future machine-generated data on the need for centralized computing environments, one needs to figure out both the potential size of the generated data *and* the fan-in factor for how much it can reasonably and inexpensively be reduced in size before reaching the central environment.

A cloud analogy

Recently the Wall Street Journal published an article titled "Cloud-Computing Kings Slow to Adapt to Own Movement."
http://www.wsj.com/articles/cloud-computing-kingpins-slow-to-adapt-to-own-movement-1438731775

In the article, it was pointed out that companies like Amazon, Google, and Microsoft weren't running all of their internal systems on their own public cloud platforms. For instance, Amazon doesn't run all its operations on AWS. To quote the article:

"The fact that Amazon still uses private servers is "ironic," said Ed Anderson, an analyst with Gartner, which advises customers on both cloud services and data center servers."

The article gives the impression that the reason for cloud-service providers not to run entirely on their own cloud is worries about reliability and security. Of course, that's not really what's going on. There is a very basic explanation. In IT, as the old saying goes, *if it ain't broke, don't fix it*. As a matter of fact, trying to fix things comes with enormous risks.

If a company is growing at around 20 percent a year, what's it annual revenue? The answer may depend on your definition. Is it based on ttm, run rate, projections for the current year, etc? But let's say, for the sake of argument, that Amazon has $100 billion in annual revenue. That's about $274 million per day. So if a move of the company's massive ecommerce infrastructure, much of it dating back to the 1990s, to AWS were to result in a single day's worth of downtime, the result would be a significant revenue loss. Nobody in his right mind would attempt such a migration of computing infrastructure willy-nilly. The MAWS (Move to AWS) project at Amazon has been going on for years and will likely go on for years to come.

Anyone with a modicum of knowledge about enterprise software knows of the perils of changing anything in a computing environment. A mere switch to newer hardware for a server can require months of preparations and testing. An upgrade to a new major release of database software for a mission-critical system might be planned years in advance and involve many months of testing before the upgraded system is ready to go live and replace the old one. Many years ago, I had an interesting conversation with a guy responsible for the Oracle databases at UBS, which had a huge number of applications running on top of Oracle. He complained that by the time they had tested all the applications against a new release of Oracle to verify that it was safe to upgrade, that new release was already so old that it was about to become desupported by Oracle. The reluctance to modify a well-running system is not the exclusive realm of paranoid Swiss bankers. Oracle makes good money from offering extended support for software releases that have been desupported. So a customer will pay extra for support for a release that is usually so mature and stable that it doesn't take much work to support it. And that's just because the customer would rather shell out the extra support money than go through the pains of an upgrade.

Which brings us to Hadoop. If cloud-service providers are slow to move their own computing infrastructure to the slightly different environment offered by their own clouds, what are the chances of existing production IT systems being moved to Hadoop, which is not just a different platform but also a completely different computing paradigm? Not great, one would think. That would essentially tie the growth opportunities of Hadoop mainly to new "Big Data" scenarios and mainly those that can't be handled by other technologies. How big those opportunities will turn out be remains to be seen. At the same time, the notion of what Hadoop is may continue to evolve. For instance, Grammarly moved from using "Amazon EMR with Hadoop and Pig" to using Spark on EC2 with S3 for storage.
http://tech.grammarly.com/blog/posts/Petabyte-Scale-Text-Processing-with-Spark.html

So Spark can run both in Hadoop clusters or in standalone mode and with a variety of input formats. If we see a tendency of Hadoop-compatible processing engines being run in environments that are not traditional Hadoop clusters, it's going to bring up the questions whether such implementations should be counted towards

the success story of Hadoop and whether they will enrich the traditional Hadoop distributers. Amazon probably doesn't care that much as long as everything is run on AWS.

Conclusion

The future success of Hadoop will depend heavily on the emergence of new sources of vast quantities of data. How that will play out remains to be seen. Data volumes will likely continue to grow significantly as has always been the case in the past. There will likely be sources for potential data growth that turn out to have been overhyped, like RFIDs were about a decade ago. (Now we have the overhyped data lake concept that has extreme security issues.) How much of the new data can be filtered, compressed, or aggregated near its sources lessening the need for large central cluster processing is something that will be a factor. As will be the retention needs for the raw new data. And the degree to which conventional means can handle mainstream needs for computing will be an important factor in the size of the Hadoop niche. A possible outcome is that Hadoop becomes a little like Artificial Intelligence. AI started out as a grand scheme of making computers intelligent like humans, a goal that is still a far-away dream. But in the process, practically useful results were achieved in subareas like robotics, natural language processing, machine learning, etc. Perhaps Hadoop will end up in a similar fashion by failing as an ambitious grand scheme but in the process, spawning some useful components.

.