Allen Wittenauer on Hadoop

An Interview

RIK FARROW

Rik is the editor of *;login:*.
rik@usenix.org

Allen Wittenauer is currently the Senior Grid Computing Architect at LinkedIn, Inc. He has been working with Hadoop with an eye toward operability for almost six years. aw@apache.org

I have wanted to run an article about Hadoop performance for a while, and my search for an expert finally paid off. Justin Sheehy, CEO of Basho, suggested that I interview Allen Wittenauer of LinkedIn.

I had naively thought that the way to improve, or at least maintain, Hadoop performance was to monitor the performance of the cluster, but Allen, with experience as both a Hadoop developer and large cluster operator, has very different thoughts on the subject.

I started out by reading the slides of a old presentation that Allen wrote called "Hadoop 24/7". Among his suggestions were to run the checks `hadoop dfsadmin -fsck` and `<t>hadoop dfsadmin -report` nightly, and to use an NFS backup of the NameNode file system image. You can find an updated version of his presentation at https://www.usenix.org/sites/default/files/login-1302-wittenauer-slides.pdf.

Rik: Let's start with some information about your background as it pertains to Hadoop. Looking at your LinkedIn references, I can see you worked for Sun, then Yahoo!, and now LinkedIn. Also, you are a Hadoop committer, which implies a lot about your ability to talk about Hadoop. Can you tell me more?

Allen: After a few years working at a software company that supported hospitals, I wanted to get involved with bigger scale problems. A friend of mine said they were hiring at Sun and would I be interested. "The network is the computer"... yes! I was lucky enough to get hired and moved up the ranks. As a result, I had the privilege of bringing order to the chaos that organic growth brings at larger and larger scales. Three such problems were building a single NIS domain to handle 100,000+ hosts spread across the entire SF Bay Area, OS provisioning on an intercontinental scale, and building a world-wide, single realm Kerberos deployment.

A layoff and a short stint at a startup later, I found myself talking to Yahoo! about Hadoop. They were interested in the same sorts of problems, but with a view toward sharing the solution with the world at large. Many companies solve these problems using home-grown tools; however, when one is trying to build a community around what was mostly (at the time) unproven technology, it is vital that you show a solution that they themselves can use. I was brought in to "rethink" Yahoo!'s operational infrastructure with an eye toward open source, what I would call "burning the house down." It was very "startup"-like, so while that's what I was hired for, I ended up doing a lot more than that and had the honor of directly impacting some of the key components in Hadoop, especially as it relates to operations.

Rik: In the earlier versions of your slides, you mentioned using `hadoop` commands nightly to check the health of the underlying HDFS. Have these suggestions changed over time?

Allen: The basics are still the same, but Hadoop has greatly matured in the past four years since this presentation was completed. While doing at least a nightly fsck to check for health is still important, there is a lot more information now available to use for basic monitoring and metrics collection via the normal Hadoop metrics plugins, JMX and JSON. In addition to CPU usage, network usage, etc., we collect a lot of that extra Hadoop data plus some additional ones such as disk service times to build a macro view of the overall system's health. While the metrics Hadoop provides tend to be extremely low level, just keeping track of values such as "tasks completed" can be useful to see if the overall grid is getting faster/slower.

One of the key points about monitoring is that you want to check the health of the service. I recommend having a tiny job that runs every 15 minutes as a canary to verify everything is working. In addition, alerting on percentage of down nodes vs. on individual node failures becomes increasingly important as you scale up. That's a common mistake when people first start working with Hadoop. We're all trained from the beginning that every machine is important because most services are fragile. If the compute nodes are properly provisioned, losing a few shouldn't matter. It's a hard concept to unlearn.

Rik: You also mentioned issues with the NameNode, which has always (for me) been the scariest part of HDFS.

Allen: From an Enterprise view of the world, the fact that single points of failure exist in the system at all is shocking. From an HPC/scientific view, not as much. I think people forget that for a long time Hadoop was being built in a "two guys in a garage" fashion to solve fairly specific problems. There wasn't a priority placed on five-9s of uptime for a system that was geared toward batch computation. Scaling up to petabytes of data and fixing performance issues related to processing that large of a data set were the priorities. It was acceptable to have, say, an hour of downtime in the case of a failure. Now that many more people are interested in using HDFS for real-time access with technologies like HBase and the ever increasing commercial interest, the focus has shifted. The 2.x branch of Apache Hadoop will include high availability of the NameNode.

Something else to consider: using something like Xen's Live Migration or another HA solution—Linux HA, SunCluster, whatever—is also an option. Administrators shouldn't be afraid of trying to apply other methods.

That said, I think too much focus is placed on this limitation. In the almost six years, 30+ grids, 30k hosts, and two companies where I've run Hadoop, the number of times where a highly available NameNode would have saved me from service downtime can be counted on one hand. Almost all cases of NameNode failures are configuration problems or bad user behavior, none of which high availability will actually help prevent from a downtime perspective. Additionally, I find it somewhat odd that no one seems to be too concerned about the lack of availability for the JobTracker. A file system with nothing running on it isn't very useful.

Rik: You have suggestions in your slides [1] for setting up the NameNode (mostly lots of memory, as that is key to performance), and recovering a NameNode using an NFS replica of the image. As I want to focus on troubleshooting performance issues, I am more interested in maintaining the health of the NameNode than

recovering it. But writing the change log to an NFS mounted file might be a performance issue. What other performance issues should people running a NameNode be aware of? Do you still recommend using NFS for the backup change log?

Allen: Until the 2.x branch is stabilized, yes, I do. At LinkedIn, we actually store our primary NFS copy on the secondary NameNode. At Yahoo!, we had some older NetApp boxes that were too small for another project that we repurposed for this task. The number of IOPS obviously scale to how busy a particular HDFS might be, but in most cases this is relatively light. Even so, it is important to make sure that the NFS server is nearby on the same network core since you don't want too much latency. Another key point is that the NameNode can work off of only one fsimage and edits file. If the primary machine has a file system failure where the image is stored, it will continue to run off of the NFS copy.

Rik: How does a sysadmin go about uncovering performance problems in a Hadoop cluster? You've said that maintaining the health of HDFS is one key, and the NameNode is another. But there may be other causes of big (noticeable) drops in performance when executing a job that gets done every day.

Allen: One of the most important actions that users can do is take a holistic view of the work they are trying to accomplish. Many, many times users will tune individual jobs, making them as fast as possible so that they can iterate during development quickly. In the process, they hit an anti-pattern in the production phase where the parallelization is too high to the point that the entire workflow is performing worse. For example, increasing reducers to the point that many small files are generated has an extremely negative network impact on the next phase of the MapReduce job that is going to read those files, either during the read of the input or in the shuffle phase.

I'm also a big fan of getting your hands dirty at the micro level. When dealing with large, scalable systems, the temptation is high to look at your metrics for the entire system and say everything looks fine. What you miss out on is that you might have one job or an internal framework or something else that has bad behavior. Averaged out, that's easily missed. But dropping down to the shell and just getting a feel for what is happening on a per-node basis is extremely helpful. Years ago, we shaved hours of processing from some very important workflows at LinkedIn by discovering a hidden bug in a commonly used internal framework. It wasn't properly caching data from an external hint file and in turn was triggering an extra I/O on one of the six disks for every record read. From the macro level, it was averaged away but dropping down, and using tools like truss, dtrace, iostat, and sar made it really stick out.

Rik: What about tunables? I hear that there are a ton of them and they almost all have an impact on performance.

Allen: Yes, there are an amazing assortment of settings that can be applied to a job. The running joke in the early days was that we'd hit a particular corner case and have to add something to the configuration to tackle that problem. Unfortunately, there is no magic bullet to know which particular tunable is the correct one to modify. This is made worse by the fact that some tunables, such as io.sort.mb, have a direct impact on how other tunables will operate. Today, the best bet is to use the job and system metrics to determine whether the setting you've changed made a difference. Usually one of the biggest mistakes people make here is to just throw more heap at the problem. In most cases, this is the wrong thing to do. So that should be the last thing that is changed.

I'm excited about tools like Duke University's Starfish project (http://www.cs.duke.edu/starfish/). We've been doing cooperative research in the hopes that exposure to "real world" conditions will improve its suggestions. In the future, this means that we will have some utilities to automatically tune workflows for nearly maximum performance.

Of course, bad algorithms will still be bad no matter how much tuning one does.

Rik: In your slides, you mention there is no real security in Hadoop, and I've seen postings to this effect. I've also read a posting by you mentioning adding Kerberos support. That means adding another dependency to Hadoop, although one that most orgs can easily support if they have AD or Samba 4, so perhaps this is not really a performance issue. But maybe I am wrong, and having an overutilized AD server or congested network link to it could slow down a Hadoop cluster..

Allen: There was a great debate about how do we secure Hadoop. For quite a while, Hadoop didn't even have the concept of permissions or even of different users. Those were only added to prevent users from accidentally deleting each others' data. There was always the thought that "we should secure the system," but other priorities prevented that from happening. Eventually, business realities forced the issue and we had the challenge of how do you secure a highly scalable service while also making it perform? We knew we didn't want the system to prompt for a password and then pass that around. Clearly we needed something that was single sign-on, the holy grail. There was a big debate around using x.509 certificates/PKI vs. Kerberos. Ultimately, the decision came down to implement SASL so that people could build their own solution if necessary, but we were going to go with built-in support for GSSAPI with Kerberos.

A big chunk of that decision was exactly as you stated: most places have Kerberos in the form of Active Directory, even if they don't know it. That takes care of the authentication portion, but now what about scale? It was decided to use a token mechanism so that individual daemons wouldn't be an inadvertent DoS against the KDCs. In this way, your Kerberos credential is used by your job client against individual services, that client gets a token, and then that token is used throughout the rest of your job's lifecycle to access other parts of the system without impacting the KDC. That takes care of overextending a potentially overutilized AD server.

Reality, however, is a bit different. Internal politics for most organizations likely make this less cut and dried. If user accounts, and therefore their AD credentials, are controlled by IT, does that same organization really want to hand out Kerberos keytab files for potentially thousands of machines that they don't control? Probably not. So what ends up happening is that companies do a one-way trust so that IT still owns the corporate infrastructure and the "other" organization (Web operations, DBAs, development, whoever) can tie their Hadoop systems to a local Kerberos infrastructure. Users still have one password (which makes them happy), the Hadoop operations team has control over their boxes (which makes them happy), and all of your data is nicely secured (which makes everyone happy). As an added bonus, this also helps balance out the performance implications as the Hadoop systems will mostly be hitting the local KDCs.

Rik: This is great information. Is there anything else you'd like to add?

Allen: Just a big thank you to the wonderful services and information that USENIX and LISA have provided over the years. I'm truly honored to be able to contribute back to the technical excellence that these organizations represent.