Scale Up or Scale Out for Graph Processing?

Jimmy Lin
University of Waterloo

This column explores a simple question: scale up or scale out for graph processing? Should we simply throw "beefier" individual multi-core, large-memory machines at graph processing tasks and focus on developing more efficient multi-threaded algorithms, or are investments in distributed graph processing frameworks and accompanying algorithms worthwhile? For rhetorical convenience, I adopt customary definitions, referring to the former as scale up and the latter as scale out. Under what circumstances should we prefer one approach over the other?

Whether we should scale up or out for graph processing is a consequential question from two perspectives: For big data practitioners, it would be desirable to develop a set of best practices that provide guidance to organizations building and deploying graph-processing capabilities. For big data researchers, these best practices translate into priorities for future work and provide a roadmap prioritizing real-world pain points.

tl;dr – I advocate scale-up solutions for graph processing as *the first thing to try*, since they are much simpler to design, implement, deploy, and maintain. If you really need distributed scale-out solutions, it means that your organization has become immensely successful. Not just "successful" but on a growth trajectory that is outpacing Moore's Law (it'll become clear what this means below). This is unlikely to be the case for most organizations, and even if you were fortunate enough to experience such explosive growth, success would likely bring commensurate resources to throw at the problem. So as long as you leave yourself enough headroom, it should be possible to build a scale-out graph processing solution just in time. The alternative is sunk costs in distributed graph processing infrastructure, which comes with huge parallelization overheads, anticipating a problem that never arrives.

It makes sense to begin by more carefully describing the scope of the problem: the type of graph processing I am referring to involves analytical queries to extract insights or to power data products. An example of the former might be clustering a large social network to infer latent communities of interest. An example of the latter might be building a graph-based recommendation system. Typically, these queries require traversing large portions of the graph where throughput, not latency, is the more important performance consideration.

Gartner defines "operational" databases as "relational and non-relational DBMS products suitable for a broad range of enterprise-level transactional applications, and DBMS products supporting interactions and observations as alternative types of transactions." I am specifically *not*

Published by the IEEE Computer Society
1089-7801/18/$33.00 ©2018 IEEE

referring to these – for example, user-facing systems that store and serve a large graph, such as Facebook's TAO.[1]

WTF AT TWITTER

Previous readers of this column know that my opinions are strongly colored by my "in the trenches" experience working full-time at Twitter from 2010 to 2012 (and subsequent part-time involvement until around 2015) on building infrastructure to support data science and platforms to power data products. For graph processing, Twitter has always been an advocate of scale-up approaches, starting with its first graph-based user recommendation project called WTF (Who-To-Follow). Gupta et al. present a more technical version of the story,[2] but here I provide a summary of relevant design rationale.

The WTF project began in the spring of 2010 as an effort to plug a gap in Twitter's product offerings. At the time, Facebook and LinkedIn already had mature user recommendation services, but Twitter had no comparable feature to suggest accounts users might be interested in following. Thus, short time to market was an important goal. From this perspective, WTF was a great success—it launched later that summer.[3]

The single biggest contributing factor to this rapid deployment, I believe, was the architectural decision to hold the *entire* graph in memory on a single server, implemented in an open-source graph processing engine called Cassovary (https://github.com/twitter/cassovary). This design very much went against the grain of conventional wisdom, which held horizontally-scalable distributed systems as the gold standard.

Why did we reach this contrarian decision? The design choice broke down into two separate considerations:

First, did the graph fit into memory? In this case, the graph of interest was the network of follow relationships on Twitter. Although many other graphs existed from which recommendations could be computed, such as those induced from engagements (replies, likes, retweets, etc.), leveraging the follow graph seemed like the obvious starting point. This was a simple and straightforward question to answer—even with the most inefficient encoding of uncompressed (source, destination) pairs, each graph edge consumes eight bytes, which translates, for example, into 64 GB for a graph with eight billion edges. Based on statistics at the time, the answer to this question was *yes*.

Second, how much "headroom" or "runway" did we have? That is, factoring in anticipated growth, how much time until we run out of memory? Based on historic data, it was possible to run projections of graph size over time under different assumptions, and from the projections it was straightforward to derive an answer. Crucially, these projections accounted for Moore's Law. That is, even if we *did nothing*, in (roughly) 18 months, Moore's Law would give us servers with double the amount of memory. We convinced ourselves that there was sufficient headroom to proceed with the scale-up design. WTF launched on servers with 72 GB RAM and, as planned, later moved to servers with 144 GB RAM.

In other words, the critical question was not "How fast is the graph growing?" but "How fast is the graph growing compared to Moore's Law."

To be clear, I'm not referring to Moore's Law literally, as it is commonly acknowledged that we are reaching the limits of Dennard scaling. Instead, I am using "Moore's Law" as a shorthand for the exponential increase in computing capabilities over time, which I am optimistic will continue for the foreseeable future (even if driven by different underlying mechanisms).

In my experience, this is a factor that most do not consider in weighing scale-up vs. scale-out solutions. Even if the graph fits into memory today, there's a nagging sense that you'll run out of memory *eventually*. But this need not be true—if the graph is growing more slowly than the increasing amounts of memory provided by future technological improvements "for free." then a scale-up architecture will remain viable *indefinitely*. Around two years ago (May 2016), Amazon introduced the X1 instance on EC2 with nearly 2TB of memory (and 128 cores to boot).[4] I wonder how many organizations have graphs larger than that?

Thus, holding everything else constant, scale-up graph processing is *indefinitely* viable unless your graph is growing faster than Moore's Law. If your graph is really growing that quickly, I say—wow, congratulations! If you started with a scale-up solution, this would indeed force you to completely replace it with a distributed scale-out solution. However, this type of explosive growth would surely attract resources sufficient for a complete architectural revamp—that's a good problem to have, since you are likely succeeding beyond anyone's wildest dreams.

Consider the alternative, which is to invest a significant amount of engineering resources up front for a scale-out solution, anticipating faster-than-Moore's Law growth that never materializes. As I'll discuss later, distributed solutions come with significant overheads—you incur a significant performance penalty at the get-go. For small organizations, especially those seeking to grow rapidly, it makes sense to prioritize effort on building products, not graph processing infrastructure. Ironically, preparing for future growth (investment in scale-out infrastructure) actually detracts from growth itself (which only comes from building useful products and services). Thus, I advocate scale-up solutions as the first thing to try.

I'll conclude this section with a bit more nuance—note that I was careful to qualify my recommendation with "holding everything else constant." There are cases, however, in which this assumption may not hold. In the case of the WTF project, the initial focus was the follow graph, represented simply as (source, destination) pairs. What if we wanted to add vertex metadata, such as properties of the user? What if we wanted to work with other types of graphs, e.g., the implicit graph defined by engagements such as likes, retweets, replies, etc.? This could be modeled by adding metadata (e.g., engagement type and timestamp) to the edges. For such a graph, a back-of-the-envelope calculation would show that trying to fit everything in memory on a single machine is unrealistic.

In truth, the broader trajectory of graph recommendation services at Twitter includes both scale-out and scale-up architectures. The original Cassovary system that powered WTF was eventually decommissioned in favor of a Hadoop-based scale-out solution.[5] Then there was a shift to a distributed scale-out real-time graph-based recommendation engine called MagicRecs,[6] followed by another scale-up fit-everyone-in-memory-on-a-single-machine real-time graph processing engine called GraphJet.[7]

The existence of these distributed scale-out systems, however, does not contradict my recommendation of scale-up as the first thing to try. At Twitter, Cassovary was always intended to be an interim solution—the goal was to get a product launched as quickly as possible, and then to leverage the experiences gained to design subsequent systems. In each of the above systems, the choice between scale-up and scale-out architectures was carefully considered and not the result of blind adherence to conventional wisdom.

Finally, holding the entire graph in memory on a single machine allowed Twitter data scientists to rapidly prototype and explore a variety of algorithms with minimal effort. In many cases, it was possible to leverage decades of research on graph algorithms to build straightforward implementations in a shared-memory setting. For many of the same algorithms, we would have needed to figure out how to build an efficient distributed implementation from scratch. Thus, starting with scale up allowed us figure out "what works" first; this then helped us to prioritize efforts in scaling out the right solution if needed.

IMITATION AS FLATTERY

It is clear that the scale-up design philosophy has had a large impact at Twitter. Even when scale-out architectures were eventually preferred, simply throwing more memory at the problem was never summarily dismissed without a careful consideration of the engineering and product merits. What about beyond Twitter?

In March 2017, seven years after the original WTF project, Pinterest published the following blog post about Pixie, their graph-based recommendation system:

> *This [the Pinterest] graph captures a huge amount of rich data from our users, and is quite large, with more than 100 billion edges and several billion nodes. Thankfully, RAM today is*

> *incredibly cheap, and big data like this is small enough to fit on readily available AWS machines. Before terabyte-scale RAM machines were available, complex distributed systems like Hadoop or Spark were needed to compute algorithms for data of this scale. Fortunately, in a way big data is actually getting smaller! Now we can load the entire graph into a single machine and traverse all of it without making any network calls. This makes real-time algorithms on densely connected graphs much easier to develop and deploy at scale, and allows us to make recommendations in real-time the moment a Pinner opens our app (instead of computing them in batch jobs the night before).* [8]

The blog post goes on to explain how Pixie powers recommendations across Pinterest and accounts for about half of all Pins saved.

It's difficult to write a more enthusiastic endorsement of the Twitter hold-the-entire-graph-on-a-single-machine design philosophy than the passage above! The observation that "big data is actually getting smaller" is exactly the Moore's Law argument I made above. Back in 2015, for the inaugural edition of this column, I posed the provocative question "Is Big Data a Transient Problem?"[9] There, I made the argument that if big data in general (not just graphs) are growing more slowly than Moore's Law, then big data processing will become easier over time as scale-up solutions become more practical. Perhaps the future is indeed unfolding this way.

So, if imitation is the sincerest form of flattery, the WTF and Cassovary engineers are definitely flattered. Once again, scale-up graph processing should be the first thing to try, and I'm heartened that other organizations are heeding this recommendation.

SURVEY SAYS

So far, this discussion has mostly focused on graph-based recommendation algorithms, from my personal experience. Looking more broadly—what do users want from graph processing software?

My colleague Semih Salihoglu recently led his students (with help by Tamer Özsu and me) on a quest to find out. They conducted a survey of graph processing systems that focused on four major questions:

- What types of graph data do users have?
- What computations do users run on their graphs?
- What software do users use to perform their computations?
- What are the major challenges users face when processing their graph data?

The survey targeted researchers (both in industry and academia) as well as users of a wide range of graph processing systems, broadly defined: graph databases (e.g., Neo4j, OrientDB), RDF engines (e.g., Virtuoso), distributed graph processing frameworks (Giraph, GraphX), query languages (Gremlin), graph libraries and visualization toolkits, and more. In total, 89 users responded to the survey.

The results were recently published in *Proceedings of the VLDB Endowment (PVLDB)*.[10] The two biggest findings could be characterized as follows:

- Scalability is unequivocally the most pressing challenge faced by the survey participants. The ability to process very large graphs efficiently appears to be the biggest limitation of existing software.
- After scalability, participants indicated visualization as their second most pressing challenge.

One of the questions in the survey asked participants about the size of their graphs. Of the 89 responses, 20 participants (8 researchers and 12 practitioners) indicated working with graphs containing more than one billion edges. Another 21 participants (8 researchers and 13 practitioners) indicated working with graphs containing between 100 million and one billion edges. In terms of

the number of vertices, 27 participants (10 researchers and 17 practitioners) indicated that their graphs contained more than 100 million vertices.

The graph size questions were formulated as discrete ranges that participants could select from, and the largest values provided were one billion edges and 100 million vertices, respectively. Therefore, the survey results are unable to tell us how much larger these graphs actually are. How many trillion-edge graphs are there? At the low end, though, these graphs still aren't *that* big—one billion edges as (source, destination) pairs translate into 8GB. With a less brain-dead encoding, such a graph easily fits into memory on a laptop today. A graph with 100 billion edges, even with a moderate amount of metadata, could comfortably fit in memory on a commodity server today.

BUT WHAT COST?

So how can we reconcile the fact that the scalability of graph processing systems is a pressing issue (according to the *PVLDB* survey) with the observation that, for example, a graph with one billion edges isn't that big, even with metadata?

I believe that Frank McSherry and his colleagues offer an answer in their 2015 HotOS paper,[11] and it's that many scale-out distributed graph processing frameworks are just terrible in terms of performance. The point they make is that "published work on big data systems has fetishized scalability as the most important feature of a distributed data processing platform." They may scale well, but it could just be because they introduce a lot of overhead that they then must overcome. McSherry et al. wondered: To what extent are these systems truly improving performance?

McSherry et al. attempted to answer this question by introducing a new metric called COST, or the Configuration that Outperforms a Single Thread. In other words, how much hardware do you have to throw at a scale-out distributed solution to achieve performance parity with a competent single-threaded implementation?

To answer this question, the researchers examined two graphs that are commonly used in many evaluations of graph processing frameworks—one with 1.5 billion edges and the other with 3.7 billion edges—and considered two sample tasks—PageRank and label propagation. McSherry et al. reported some surprising results. One key finding was that none of the evaluated distributed graph processing frameworks, running on 128 cores, consistently beats a single-threaded implementation running on a commodity laptop!

Although the criticisms of McSherry et al. are directed primarily at academic publications, their point also applies to the cottage industry of companies peddling distributed graph processing solutions. On the Web, you'll come across countless blog posts and white papers touting the scalability of a particular company's solution. I'd like to know the COST.

The fundamental, unavoidable truth is that once you've made the decision to abandon holding the graph on a single machine, you incur the orders-of-magnitude higher latencies that come with network communications. You've already dug yourself a hole—for the sake of horizontal scalability—that the architecture needs to climb out of. Twitter's more recent distributed graph engine MagicRecs[6] alleviates this issue by cleverly partitioning the graph such that all operations are node-local, i.e., there is no cross-node exchange of graph adjacency lists. However, the design is highly application specific, and hidden in its architecture is a part of the graph that must be fully replicated across all nodes in order to make the algorithm work.

Distributed scale-out architectures are complex. Just a few of the issues:

- Queries need to be routed to appropriate partition servers and results then need to be aggregated;
- There must be some service discovery mechanism for the query broker to discover the partition servers;
- There must be some health-check and self-healing mechanism to ensure that a minimum number of replicas for each partition is available at all times;

- Deployment is more complex, and rolling restarts with proper safeguards and canary tests are much more difficult to get right;
- And the list goes on.

In contrast, by holding the entire graph in memory, we just need a fleet of identical machines provisioned for the requisite query throughput, with a load balancer in front. This is a far simpler architecture, so a lot less can go wrong.

I believe one major reason that organizations are experiencing scalability pains processing "large" graphs is their default choice to adopt scale-out solutions, which McSherry et al. showed are terrible. Of course, this is not a blanket statement that *all* distributed graph-processing frameworks are terrible, just that there are a lot of them out there. Most distributed frameworks never dig themselves out of the hole they started in.

CONCLUSIONS

If there's a solid case for scale up as a viable alternative to scale out for graph processing, then why haven't more organizations considered this alternative? I propose two explanations.

First, I believe that at a fundamental level, humans have difficulty reasoning about exponential growth and a baseline characterized by exponential growth. For example, when we hear about an investment fund manager who has achieved impressive returns over the last *n* years, most people's initial reaction is *not* to ask about the return of a well-diversified index fund over the same time period.

Second, as another example, a cloud provider such as Amazon Web Services can raise its profit margins over time by simply reducing its per-unit cost slower than the dropping price of the underlying hardware. Per-unit resource costs are dropping due to technological progress, and of course Amazon should pass the savings on to customers, right? However, I suspect that most customers are happy with constant per-unit resource costs and are planning future use around that assumption. When Amazon lowers prices, it comes as a pleasant surprise, whereas the correct reaction should be to question if the prices have been lowered sufficiently.

The same issues are at play with scale-up solutions, in that there is this visceral unease in your gut—the models and projections seem reasonable, but is it *really* going to work? It's difficult to think in terms of exponential growth. Even as the purveyor of this scale-up advice, I distinctly remember expressing amazement two years ago with the introduction of the EC2 X1 instance with ~2TB RAM. If I had truly internalized my own advice, the reaction should have actually been, "It's about time!"

I believe another consideration is the fact that distributed scale-out solutions have been codified in our collective engineering psyches as the "one true way" and that it has become the safe default. This is the modern equivalent of "Nobody ever got fired for buying IBM." In truth, many organizations adopting technological solutions never go through careful considerations of the engineering, product, and resource tradeoffs. In many cases, an executive with purchasing power is simply checking off buzzwords ("Horizontal scalability!" "No single point of failure!" "Cloud enabled!" "Integrates with the data lake!").

As a concrete example, a number of years ago I asked some Cloudera colleagues why so much effort was being expended in developing HDFS namenode failover mechanisms, some of which were really kludgy. In my view, the amount of engineering resources devoted to tackling the problem was disproportionally more than the reality of the problem—I thought there were far more pressing pain points in the Hadoop ecosystem. The response I got was essentially that pointy-hair bosses who write the checks need to make sure that the "highly-available, no SPOF (single point of failure)" square on their buzzword bingo cards was covered.

Deploying an ill-conceived distributed scale-out graph processing system is dangerous. First, as McSherry et al. showed,[11] organizations need to throw a lot of resources at digging themselves out of the hole that those very systems created in the first place. At the end of the day, significant resources have been sunk anticipating growth that may never actually materialize. There are of

course organizations (for example, Facebook and Google) that genuinely need scale-out graph processing, but that isn't the common case by far. Better to start with scale-up solutions for graph processing.

ACKNOWLEDGEMENTS

I'd like to thank Siddhartha Sahu, Semih Salihoglu, and Aneesh Sharma for comments on earlier drafts of this column.

REFERENCES

1. N. Bronson et al., "TAO: Facebook's Distributed Data Store for the Social Graph," *Proceedings of the 2013 USENIX Annual Technical Conference* (USENIX ATC 13), 2013.
2. P. Gupta et al., "WTF: The Who to Follow Service at Twitter," *Proceedings of the 22th International World Wide Web Conference* (WWW 13), 2013.
3. "Discovering Who To Follow," blog, Twitter, 2010; https://blog.twitter.com/official/en_us/a/2010/discovering-who-to-follow.html.
4. J. Barr, "X1 Instances for EC2—Ready for Your Memory-Intensive Workloads," *AWS News Blog*, blog, Amazon, 2016; https://aws.amazon.com/blogs/aws/x1-instances-for-ec2-ready-for-your-memory-intensive-workloads/.
5. A. Goel et al., "Discovering Similar Users on Twitter," *Proceedings of the Eleventh Workshop on Mining and Learning with Graphs*, 2013.
6. P. Gupta et al., "Real-Time Twitter Recommendation: Online Motif Detection in Large Dynamic Graphs," *Proceedings of the VLDB Endowment*, vol. 7, no. 13, 2014, pp. 1379–1380.
7. A. Sharma et al., "GraphJet: Real-Time Content Recommendations at Twitter," *Proceedings of the VLDB Endowment*, vol. 9, no. 13, 2016, pp. 1281–1292.
8. P. Eksombatchai and M. Ulrich, "Introducing Pixie, an advanced graph-based recommendation system,"; https://medium.com/@Pinterest_Engineering/introducing-pixie-an-advanced-graph-based-recommendation-system-e7b4229b664b.
9. J. Lin, "Is Big Data a Transient Problem?" *IEEE Internet Computing*, vol. 19, no. 5, 2015.
10. S. Sahu et al., "The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing," *Proceedings of the VLDB Endowment*, vol. 11, no. 4, 2017, pp. 420–431.
11. F. McSherry, M. Isard, and D.G. Murray, "Scalability! But at what COST?," *Proceedings of the 15th Workshop on Hot Topics in Operating Systems* (HotOS XV), 2015.

ABOUT THE AUTHOR

Jimmy Lin is the David R. Cheriton Chair in the David R. Cheriton School of Computer Science at the University of Waterloo. His research interests lie at the intersection of information retrieval and natural language processing, with a particular focus on big data and large-scale distributed infrastructure for text processing. Lin has a PhD in electrical engineering and computer science from Massachusetts Institute of Technology. Contact him at jimmylin@uwaterloo.ca.