Performance in the Spotlight

Adrian Colyer
Accel
acolyer@accel.com

ABSTRACT

Performance in its various guises features prominently in research evaluations, and rightly so. Without adequate performance a system is not fit for purpose. That doesn't necessarily mean we should pursue performance at all costs though. In this talk we'll explore a variety of additional evaluation criteria, with a focus on those that are most important to practitioners, and ask whether or not considering them can open up interesting avenues of research.

PVLDB Reference Format:
Adrian Colyer. Performance in the Spotlight. *PVLDB*, 12(12) : 2287-2289, 2019.
DOI: https://doi.org/10.14778/3352063.3352144

1. INTRODUCTION

VLDB is "a premier international forum for data management and database researchers, vendors, practitioners, application developers, and users" [1]. It's the Formula 1 of data systems, where race cars carefully prepared by teams of researchers line up to compete around the great racetracks: YCSB [2], TPC-C [3], TPC-H [4], and friends. To the cars with the fastest lap times and the best performance over race distance go the spoils of victory. A champagne spray on the podium, fame and celebrity, time in the spotlight. Or a published paper at VLDB anyway! I'm exaggerating for effect of course, but it's certainly true that in the papers I read, the vast majority of evaluations are focused very heavily on performance.

As fabulous as those Formula 1 cars are though, you don't see many of them on the road. Even if we equalised the purchase price, you wouldn't want one as a daily driver. You need an entire team to operate one - a pit crew in your driveway! Tyre life is terrible, and fuel economy isn't great either. Service intervals are incredibly short. And then there's the user experience: the ride is uncomfortable, the cabin is cramped, the controls are confusing. Good luck fitting in even a small bag of groceries on the way back from the shops.

So when it comes to cars at least, performance obviously does matter but there are clearly a number of other factors that also have a significant influence on fitness-for-purpose away from the racetrack. Is the same thing true for database systems? Sometimes absolute performance really is the prime consideration - either because you're competing on latency or because of the sheer volume you need to process. In my experience though it's much more common to have a performance *bar* set, e.g., we need to be able to handle at least this many requests per second, and respond within *n* milliseconds. Once the bar is met, attention turns to other factors. We don't always want to have to squeeze into a six-point safety harness and put on a crash helmet just to take our data for a spin!

Proceedings of the VLDB Endowment, Vol. 12, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3352063.3352144

2. EVALUATION CRITERIA

What other factors are considered when making technology selections? Scalability certainly, as in "will I hit a wall as my business grows?" This aspect of scalability does tend to get good coverage in research paper evaluations. We're also increasingly interested in scaling *down* as well as up. Under periods of very light or no load does the system reduce its resource consumption accordingly? How rapidly can the system adjust to changes in load? Resource consumption, especially on cloud platforms, is a proxy for cost, and we want systems that support usage-based pricing. So there's a big difference between a system that offers very high levels of performance through static (over-)provisioning and hence a constant high baseline cost, and one that can *attain* very high levels of performance on demand but can also scale costs down to near zero in the absence of demand.

Performance, scalability, and "size of my cloud provider bill" cost are all reasonably easy to quantify, and hence fit naturally into evaluation sections. If we were to write an objective function for the technology selection criteria I most commonly observe in practice though, these would be constraints, not the variables to be maximised or minimised. Such an objective function would be something like "subject to performance within this ballpark, costs no greater than *x*, and with some headroom to grow, *minimise the human costs of developing solutions for, and operating, the platform*." Developer experience and operator experience are all-important, and very often absent from evaluations. Of the two, in a world where developers often have the power to make technology selections, it shouldn't be a great surprise that I frequently observe developer experience playing a very significant part in determining marketplace success.

There is a gap between the typically performance-oriented focus in research evaluations, and the criteria used in practitioner technology selection evaluations. Does this gap matter? I think it does for two reasons: firstly, it puts us at the mercy of Goodhart's law [5], and secondly it means a whole set of interesting research questions are getting less attention than they otherwise might. The streetlight effect [6] explains why we tend to put more emphasis on measuring criteria that are easy to measure and quantify, at the expense of criteria such as developer and operator experience which are more subjective and harder to quantify. Goodhart's law, "*when a measure becomes a target it ceases to be a good measure*," then kicks in and if we are not careful will cause us to optimise for performance at all costs. Even if the result is, for example, a highly-strung system with very narrow operating margins.

2.1 Developer Experience

If developer experience were an orthogonal concern, chiefly addressed by good documentation, tutorials, and samples, then it would be much less interesting from a research perspective. However, we know that system design truly does have a big impact on the developer experience. Consider something as simple as the way we connect to a datastore. Connection-based protocols made sense in a world of static deployments, three-tier web applications, and connection pools. But they don't work nearly so well for cloud-native applications with lots of fine-grained auto-scaling components. A connectionless protocol is a much better fit here.

Then of course there is the consistency model. I have come to the conclusion that for the vast majority of developers, the mental model they have when developing an application is strict serializability. That is, they look at the lines of code in front of them and reason as if there were no other concurrent activity possible. This mental model holds regardless of the actual underlying consistency model(s) supported by the system. So the set of anomalies that can occur given any weaker consistency model also amount to a set of ways developers can get caught out! A classic example of this is reported by Google in "Spanner, becoming a SQL system" [7], in which the authors describe Spanner's evolution towards a fuller database-like experience over time because developers at Google "*found it difficult to build applications without a strong schema system, cross-row transactions, consistent replication, and a powerful query language*."

The data model and the programming model layered on top of it have a lot to say about what's easy to express, what's possible to express, and also what classes of developer mistakes are possible. See "Understanding real-world concurrency bugs in Go" for a recent discussion of these issues in the context of the design of the Go programming language [8] and Van Roy's "Programming paradigms for dummies" for a broader programming language survey [9]. Bloom [10] is a good example of the creative possibilities at the intersection of programming paradigms and data paradigms.

Developers probably aren't going to get strict serializability (arguably they've never really had it! [11]), although we may be able to get close [12]. What happens when the system fails to meet their implicit expectations? Are application invariants silently broken with inconsistencies lying in wait as-yet undetected? Memories, guesses, and apologies [13] gives us a way of thinking about building applications in the real world, but how do we know when we need to apologise? What would a programming model be like that treated potential anomalies like checked exceptions for example?? But maybe not every potential anomaly is an issue for a given workload: if we could rely on developers to accurately articulate the actual invariants for a workload then we could focus reparation efforts where it really matters [14], and understanding invariants at the application level can also be a key to unlocking performance by avoiding coordination where we know we don't need it [15]. Is this a realistic expectation though? And how could we assist developers in getting it right?

Beyond correctness, there are other important components to developer experience too. Predictability is one. Can small changes to the application or data model have big impacts on performance? Is it easy to predict when this will happen or can a seemingly innocuous change catch you out? The ability to easily diagnose and debug problems is another. When an output isn't what you expect, can you determine why? Can the provenance of data easily be captured and explained?

I expect provenance to be an increasingly important concern for future data systems and applications, right alongside purpose, permission, and privacy. At point of capture we obtain consent to use data for a given purpose or purposes. We need to track provenance as we join, project, and aggregate, which informs the permissions we grant to see or use the results and to preserve privacy expectations.

2.2 Operator Experience

System design clearly has a big impact on operator experience too. How does the operational overhead scale as the system scales? Under what circumstances is operator intervention required? How many configuration parameters and parameter values does the system expose? Since parameter spaces can quickly become vast, it probably makes sense to compare these across systems on a log-scale. Fewer is generally better. Does the system offer generally good performance across a broad sub-space of parameters, with predictable and gentle gradients as parameter values are changed? Or is tuning a dark art with sharp performance peaks and steep gradients?

How do hotspots and failures propagate across the system, and how long does it take to recover from them? Gan et al.'s recently published microservices benchmark suite, DeathStarBench [16], illustrates nicely how architectural decisions, such as the way responsibilities are divided amongst distributed components, can impact this. Does a database system play nice when placed in a broader architectural context? Are tail latencies tightly bounded?

Just like developers, when things go wrong operators value the ability to easily explain the observed behaviour and troubleshoot the system.

3. STANDING UNDER THE STREETLIGHT

The traditional story that gave its name to the streetlight effect goes as follows:

A policeman sees a drunk man searching for something under a streetlight and asks what the drunk has lost. He says he lost his keys and they both look under the streetlight together. After a few minutes the policeman asks if he is sure he lost them here, and the drunk replies, no, and that he lost them in the park. The policeman asks why he is searching here, and the drunk replies, "this is where the light is." [6]

A researcher is trying to build a better database system. A colleague comes along and asks where the big opportunities are. After the researcher explains, the colleague asks why therefore the researcher is focusing so hard on pushing the frontiers of performance. And the researcher replies, "this is where the spotlight is."

4. ACKNOWLEDGMENTS

With thanks to Pat Helland and Wolfgang Lehner for encouraging me to submit a talk and for their feedback, help, and guidance in getting me through the process.

5. REFERENCES

[1] VLDB 2019: Overview, 2019. http://vldb.org/2019

[2] Cooper, B., Silberstein, A., Tam, E., Ramakrishnan, R. and Sears, R. Benchmarking Cloud Serving Systems with YCSB. In *ACM Symposium on Cloud Computing (SoCC '10)* (Indianapolis, IN, USA, 2010).

[3] TPC-C On-Line Transaction Processing Benchmark. http://www.tpc.org/tpcc/default.asp

[4] TPC-H Decision Support Benchmark. http://www.tpc.org/tpch/default.asp

[5] Goodhart's Law. https://en.wikipedia.org/wiki/Goodhart%27s_law. Accessed May 2019

[6] Streetlight Effect. https://en.wikipedia.org/wiki/Streetlight_effect. Accessed May 2019

[7] Bacon, D., Bales, N., Bruno, N., Cooper, B., Dickinson, A., Fikes, A., Fraser, C., Gubarev, A., Joshi, M., Kogan, E., Lloyd, A., Melnik, S., Rao, R., Shue, D., Taylor, C., van der Holst, M. and Woodford, D. Spanner: Becoming a SQL System. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 331-343, 2017.

[8] Tu, T., Liu, X., Song, L. and Zhang, Y. Understanding Real-World Concurrency Bugs in Go. In *Proceedings of 2019 Architectural Support for Programming Languages and Operating Systems (ASPLOS'19)*, 2019

[9] Van Roy, P. Programming Paradigms for Dummies: What Every Programmer Should Know. In *New Computational Paradigms for Computer Music*, 2009.

[10] Alvaro, P., Conway, N., Hellerstein, J., and Marczak, W. Consistency analysis in Bloom: a CALM and Collected Approach. In *CIDR*, 2011

[11] Fekete, A., Goldrei, S., and Asenjo, J. Quantifying Isolation Anomalies. *PVLDB*, 2(1):467-478, 2009.

[12] Bailis, P., Davidson, A., Fekete, A., Ghodsi, A. Hellerstein, J. and Stoica, I. Highly available transactions: Virtues and limitations. *PVLDB*, 7(3):181-192, 2013.

[13] Helland, P. and Campbell, D. Building on quicksand In *CIDR*, 2009.

[14] Balegas, V., Duarte, S., Ferreira, C., Rodrigues, R., Preguiça, N., Najafzadeh, M. and Shapiro, M. Putting consistency back into eventual consistency. In *Proceedings of the Tenth European Conference on Computer Systems*, ACM, 2015.

[15] Bailis, P., Fekete, A. Franklin, M., Ghodsi, A., Hellerstein, J., and Stoica, I. Coordination avoidance in database systems. *PVLDB*, 8(3):185-196, 2014.

[16] Gan, Y., Zhang, Y., Cheng D., Shetty, A., Rathi, P., Katarki, N., Bruno, A., Hu, J., Ritchken, B., Jackson, B., Hu, K., Pancholi, M., He, Y., Clancy, B., Colen, C., Wen, F., Leung, C., Wang, S., Zaruvinsky, L., Espinosa, M., Lin, R., Liu, Z., Padilla, J. and Delimitrou, C. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems. In *Proceedings of the 2019 Architectural Support for Programming Languages and Operating Systems (ASPLOS'19)*, ACM, 2019.