Computing without Servers, V8, Rocket Ships, and Other Batsh*t Crazy Ideas in Data Systems

Jimmy Lin
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
jimmylin@uwaterloo.ca

1 TWO TRENDS

Computing is simultaneously becoming more centralized and more distributed. A relatively small handful of companies are building increasingly-large datacenters to power centralized cloud services at mind-boggling scales. At the other end of the spectrum, personal computing devices continue to proliferate: Although mobile phones and tablets are reaching saturation, smart personal devices and home gadgets, not to mention other devices falling under the Internet of Things umbrella, are increasingly ubiquitous.

My talk discusses these two trends in the context of information retrieval systems, and more broadly, data systems that store, process, analyze, search, and manipulate large amounts of data. I share some research by my colleagues, students, and myself exploring implications of these trends that are "unconventional", "off the beaten path", or simply batsh*t crazy.

2 TO THE CLOUD!

The idea that the "datacenter is the computer" is about a decade old [2, 17] but today there remains no consensus on what the "datacenter operating system" should look like or what the "basic unit of computing" should be. In the beginning, clouds provided the abstraction of virtual machines, from which a customer could "wire together" clusters for various tasks—massively-parallel data processing using Hadoop was the first killer application. Over time, other abstractions emerged: containers are replacing virtual machines as the basic building block, and we are witnessing a proliferation of "everything as a service". Just a few examples include storage as a service, database as a service, search as a service, messaging as a service, logging as a service, analytics as a service, and machine learning as a service. We even see examples of meta-services, for example, configuration as a service, scaling as a service, and my favorite: service fabric as a service. Microservice architectures are all the rage these days!

One way to think about the proliferation of these cloud services is in terms of the disaggregation of computing capabilities. Of course, after provisioning one or more virtual machines (or containers), I can deploy anything I want, say, Lucene/Solr or MySQL. However, if my ultimate goal is to enjoy the capabilities offered by these applications (full-text search and SQL querying, respectively), then the virtual machine (or container, even) is too low level an abstraction. I don't want to worry about scaling out and partitioning data across multiple nodes, installing software updates, patching security vulnerabilities, and dealing with a host of other administrative headaches. Instead, I'll take advantage of higher-level offerings (and willingly pay a premium over raw virtual machines): this is the value proposition of search as a service or database as a service. Taking this idea to one extreme, we arrive at serverless computing [1, 18]. Specifically, so-called "function as a service" offerings such as AWS Lambda and Azure Functions allow developers to write blocks of code with well-defined entry and exit points, delegating all aspects of execution to the cloud provider. Typically, these blocks of code are stateless, reading from and writing to various "state as a service" abstractions (databases, message queues, persistent stores, etc.).

Think of "disembodied" functions as the basic unit of computation: I just want to run this block of code and not have to worry about anything else. Behind the curtain, the cloud provider is orchestrating the provisioning of containers, load balancing across a fleet of servers as requests pile up, etc. but as the developer, none of this is my concern. Standard serverless deployments are characterized by asynchronous, loosely-coupled, and event-driven processes that touch relatively small amounts of data [8]. Consider a canonical example that Amazon describes: an image processing pipeline such that when the user uploads an image to a website, it is placed in an S3 bucket, which then triggers a Lambda to perform thumbnail generation. The Lambda may then enqueue a message that triggers further downstream processing.

What does this have to do with information retrieval and data systems? We have prototyped and explored a few unconventional applications of serverless architectures. For example, it is possible to build a serverless search engine [4]: "state" in this context comprises the postings lists, and query evaluation (i.e., traversal of the postings) become "stateless" functions. A serverless deployment of neural networks for inference [20] is also possible, following a similar design pattern. Most recently, we've ported the distributed Spark analytics platform to a serverless architecture in a system called Flint [10], which is a PySpark-compatible execution engine. With Flint, the analytics experience is indistinguishable from that of cluster-based Spark—the developer fires up a Python shell (appropriately configured) and begins doing data science. We have demonstrated serverless data analytics for simple scan-and-aggregate queries on hundreds of gigabytes of data. There are, of course, shortcomings with each of these prototypes, but they represent novel ways of decomposing monolithic systems in this brave, new, everything-as-a-service world.

In a nutshell, serverless architectures enable computing without servers in the sense that they raise the level of abstraction to *computations*, isolating the developer from mundane details of execution

DESIRES 2018, August 2018, Bertinoro, Italy

on physical machines. The developer doesn't need to manage server instances as load scales up or down.

3 TO THE EDGE!

Although computing resources are increasingly centralized in the cloud, end users ultimately access the rich capabilities it provides though personal devices. For convenience, I'll collectively refer to these as "the edge". These devices are heterogeneous both in terms of hardware and software, and increasingly *not* the user's laptop. Most of the companies building clouds would like to keep these personal devices as dumb as possible because their business models revolve around gathering user data, and local devices potentially "obscure the view".

The truth, however, is that personal devices pack a non-trivial amount of computing power these days. What are interesting things that we can do with this power?

Let's start with the web browser, which is the focal point of convergence in terms of accessing cloud capabilities: email, documents, spreadsheets, presentations, and tons of other applications are routinely accessed by hundreds of millions of users all around the world via web browsers. The vision of Chrome OS is that the browser *is* the operating system. With the advent of browser-based IDEs, notebooks for data science (e.g., Jupyter), and online collaboration tools, it is increasingly viable that even a software developer or a data scientist may never need to leave the browser. However, in nearly all deployments, the browser is relegated to a relatively-dumb rendering endpoint—computations related to user interface elements may be handled on device, but all non-trivial processing, storage, and data manipulation occurs in the cloud.

This approach vastly under-utilizes the tremendous processing capabilities available at the edge: web browsers today embed powerful JavaScript engines capable of powering online multi-player games, rendering impressive 3D scenes, supporting complex, interactive visualizations, and even running first-person shooters. These applications take advantage of HTML5 standards such as WebGL, WebSocket, and IndexedDB, and therefore do not require additional plug-ins (unlike, for example, Flash).

Can JavaScript engines in the browser be applied in interesting ways? Indeed, yes! I've shown that it is possible to build a JavaScript search engine that runs completely self-contained in the browser [12]—this includes parsing documents, building the inverted index, gathering terms statistics for scoring, and performing query evaluation. Once a collection has been downloaded, all subsequent interactions (searching, browsing, etc.) no longer require an internet connection. Of course, performance and scalability lags far behind a "real" search engine running natively, but the prototype is certainly usable.

Next up, we developed Afterburner [6], a prototype analytical RDBMS implemented in JavaScript that runs completely in the browser. In other words, your webpage has the SQL engine embedded inside it! A few clever (at least in my opinion) technical tricks were necessary to make everything work: in-memory columnar storage using typed arrays and query compilation into asm.js [7], but the upshot is that the performance of our prototype approaches the performance of an existing columnar database and modern query compilation techniques running natively.

Search engine in JavaScript? *Check*. Database in JavaScript? *Check*. The next obvious question: What about neural networks? Of course! In fact, others have long had this idea, the most recent incarnation of which is TensorFlow.js,[1] a JavaScript library for training and deploying ML models in the browser and on Node.js. For text processing applications, there was one more detail we had to iron out related to storing word embedding vectors[2] [11], but otherwise it works as expected. As our demo application, we run inference on a simple convolutional neural network for sentiment analysis [9] in the browser.

These three applications are technically interesting since they illustrate how far we can push JavaScript, but I argue that they are more than curiosities. Take the in-browser search engine example: Nervous about search engine companies mining your query logs for potential illnesses you (or your loved ones) might have? For example, the work of White and Horvitz [22] does exactly that. It's a scary prospect that companies have the technology to build demographic profiles (for ad targeting) that include features such as "gender", "age", "interest" (the obvious ones) ... and now, "may have lung cancer". What about potential data leakage (or even outright selling of data) to insurance companies? One can combat this by downloading a pre-packaged, curated collection of health information, and then searching it locally. There are no query logs for any corporation to collect and no one would know what you were searching for. The same selling point applies to neural network inference—no one needs to know what you fed into the neural network... could be your deepest, darkest secrets.

An immediate and obvious objection to this line of thinking is that local execution doesn't *actually* require JavaScript—any downloadable executable will do. However, JavaScript-based deployments have the advantage of seamless integration in a browser-centric world: your self-contained search engine, database, neural network... they're all just webpages.[3] The ubiquity of JavaScript is its greatest strength—it's probably defensible to argue that JavaScript is the most widely-deployed and easily-accessible platform in the world. Targeting JavaScript means that an application will run anywhere with a browser: a mobile phone, a tablet, even the connected toaster of tomorrow. For example, in Liang et al. [11] we compared neural network inference performance with several configurations: PyTorch on Linux, a browser on Linux, and the same browser on iPads, iPhones, and Android phones. In fact, at the conferences where we demonstrated our prototypes, we used tablets as a more convenient form factor.

Beyond privacy, JavaScript deployments open up opportunities for new execution models in unifying, and then blurring, the distinction between client-side and server-side processing. In fact, that was one of the original motivation for Node.js—developers were writing piles of JavaScript code running in the browser (since there were no viable alternatives), so why not just run JavaScript on the server side also? For databases, the idea of split execution (running part of the query plan on the client side) goes back decades [5], but JavaScript allows for some interesting new possibilities [7]. For

[1] https://js.tensorflow.org/

[2] Which can be quite large, and hence not directly embeddable in a webpage.

[3] As an aside, those up-to-date with web technologies might bring up WebAssembly. JavaScript and WebAssembly are tightly integrated and designed to interoperate in a way that I don't think the latter fundamentally alters any of my arguments.

neural networks, it would make sense to run latency-sensitive inference (for example, typeahead prediction with smart keyboard apps) on the client side, rather than to depend on shuttling everything to the cloud and trying to hide latency in other ways. Or better yet, some neural networks are quite deep, right? Let's run some of the layers locally and some of the layers in the cloud [19], and we can decide when to quit if the predicted accuracy is good enough. Wait, that's just like early exits in multi-stage ranking architectures? (I'll let Shane tell you all about them in his talk.)

So, the future is… JavaScript? Once we get beyond the fact that JavaScript is an undeniably shitty language on which to build an interlingual execution platform, there is at least some so-crazy-it-might-actually-work appeal to this idea. Or at least, why not, and worth looking into. Additional musings along these lines are provided in a column [15] for interested readers.

4 TO MARS!

The tension between centralized and distributed organizations of computation manifests in the tug of war between what happens in the cloud and what happens at the edge. Google's vision, for example, is that everything happens in the cloud, as operationalized in Chrome OS. The business model of most internet companies aligns most closely with this viewpoint, since they want to gather as much behavioral data about users as possible. Interestingly, Apple is perhaps the exception here, since it wants you to spend a fortune buying their physical devices, and that requires convincing you of cool things you can do directly on those devices. Regardless, the cloud-centric vision seems pretty clear, but what might the edge-centric end of the spectrum look like?

Our (initial and incomplete) exploration along these lines is a system called Prizm [16], a prototype lifelogging device that comprehensively records a user's web activity. The physical device is a wireless access point deployed on a Raspberry Pi that provides a substitute for the user's normal wireless access point. Prizm proxies all HTTP(S) requests from devices connected to it and records all activity it observes. The original intent of the system was to provide users with an overview of their "information diet", in the same way that apps help users track what they eat. With Prizm, the user can answer questions such as: How much time have I spent on Facebook today? How many YouTube videos have I watched this week? How many Google queries have I issued last month?

In short, Prizm is a gatekeeper sitting between a user's edge devices and the outside world. The yet-to-be-realized vision is that this gatekeeper would manage traffic in an intelligent way. In addition to metadata about all web activity, the device could store the actual content—providing a means of personal web archiving. Think of browsing histories on steroids.

Why? Previous studies have shown that a significant fraction of users' search behavior on the web consists of "refinding" [21], or searching for pages they had encountered before—in which case, why not simply return the previously-stored content? Storage is cheap and plentiful, even on small devices: so why not store a copy of Wikipedia [13], or even a local web collection? If the proxy believes that the user's query can be answered locally without going to the outside world, why not? In other words, Prizm becomes the user's intelligent agent for mediating interactions with the cloud—it could even, at the user's request, generate fake requests to confuse various internet services (call it "log stuffing").

Let's push this idea of cloud/edge separation even further... and imagine the edge on Mars! While Martian colonies may be a decade or more in the future, plans are being actively developed with the public support of luminaries such as Elon Musk, Jeff Bezos, and Edwin "Buzz" Aldrin (the second person to walk on the Moon). As opposed to a traditional Apollo-style there-and-back-again mission, many are planning for permanent settlement, with colonists potentially living out the remainder of their lives on Mars. While the idea of permanent settlement may seem like science fiction to some, there are substantial cost savings from permanent colonization since fuel and other resources for immediate return would not be required. According to proponents, becoming a multi-planet species is an inevitable development in the history of our civilization.

In this context, we explored a simple question [3, 14]: How would we provide a high-quality search experience on Mars, where the fundamental physical limit is speed-of-light propagation delays on the order of tens of minutes? On Earth, users are accustomed to nearly instantaneous responses from web services. Is it possible to overcome orders-of-magnitude longer latency to provide a tolerable user experience on Mars? This scenario shares many similarities with the Prizm concept—the fundamental tradeoff is between latency (waiting for Earth-based responses) and bandwidth (pre-fetching or caching data on Mars). We imagine starting the process with a big sneakernet delivery of a cache of the web to Mars on a cargo rocket (petabytes literally hurtling through space). Once the colonists arrive, a central proxy (i.e., an intelligent agent) would manage precious bandwidth in Earth-bound transmissions, quite similar to what Prizm might do (albeit with different tradeoffs).

Is search from Mars idle "blue sky" (or shall we say, "red sky") thinking? We intended for this research problem to be inspirational—even if one is not convinced by the premise of Mars colonization, there are Earth-based scenarios such as searching from rural villages in India or from the Canadian high arctic that share similar constraints, and thus solutions to these challenges can have impact much closer on Earth.

5 TO THE FUTURE!

Thomas Henry Huxley famously said, "It is the customary fate of new truths to begin as heresies and to end as superstitions." That is, ideas that seem crazy today might become self-evident and unquestioned in the future. However, Carl Sagan quipped, "They laughed at Columbus, they laughed at Fulton, they laughed at the Wright Brothers." Following up: "But they also laughed at Bozo the Clown. Being laughed at does not mean you are right." Perhaps I'm Bozo the Clown, and these ideas are just plain stupid. I don't know. I think not, but only time will tell!

ACKNOWLEDGMENTS

I am grateful for all the wonderful colleagues and amazing students and postdocs I've been fortunate enough to have collaborated with on the research described here. None of this work would have been possible without the generous support of many organizations over the years, and the freedom afforded by an academic position to pursue batsh*t crazy ideas.

REFERENCES

[1] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, and Philippe Suter. 2017. Serverless Computing: Current Trends and Open Problems. *arXiv:1706.03178v1.*

[2] Luiz André Barroso and Urs Hölzle. 2009. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines.* Morgan & Claypool Publishers.

[3] Charles L. A. Clarke, Gordon V. Cormack, Jimmy Lin, and Adam Roegiest. 2017. Ten Blue Links on Mars. In *Proceedings of the 26th International World Wide Web Conference (WWW 2017).* Perth, Australia, 273–281.

[4] Matt Crane and Jimmy Lin. 2017. An Exploration of Serverless Architectures for Information Retrieval. In *Proceedings of the 3rd ACM International Conference on the Theory of Information Retrieval (ICTIR 2017).* Amsterdam, The Netherlands, 241–244.

[5] Michael J. Franklin, Björn Thór Jónsson, and Donald Kossmann. 1996. Performance Tradeoffs for Client-Server Query Processing. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data.* Montreal, Quebec, Canada, 149–160.

[6] Kareem El Gebaly and Jimmy Lin. 2017. In-Browser Interactive SQL Analytics with Afterburner. In *Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data (SIGMOD 2017).* Chicago, Illinois, 1623–1626.

[7] Kareem El Gebaly and Jimmy Lin. 2018. In-Browser Split-Execution Support for Interactive Analytics in the Cloud. *arXiv:1804.08822v1.*

[8] Scott Hendrickson, Stephen Sturdevant, Tyler Harter, Venkateshwaran Venkataramani, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2016. Serverless Computation with OpenLambda. In *Proceedings of the 8th USENIX Conference on Hot Topics in Cloud Computing (HotCloud'16).*

[9] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014).* Doha, Qatar, 1746–1751.

[10] Youngbin Kim and Jimmy Lin. 2018. Serverless Data Analytics with Flint. In *Proceedings of the Third International Workshop on Serverless Computing (WoSC 2018).* San Francisco, California.

[11] Yiyun Liang, Zhucheng Tu, Laetitia Huang, and Jimmy Lin. 2018. CNNs for NLP in the Browser: Client-Side Deployment and Visualization Opportunities. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations.* New Orleans, Louisiana, 61–65.

[12] Jimmy Lin. 2015. Building a Self-Contained Search Engine in the Browser. In *Proceedings of the ACM International Conference on the Theory of Information Retrieval (ICTIR 2015).* Northampton, Massachusetts, 309–312.

[13] Jimmy Lin. 2015. The Sum of All Human Knowledge in Your Pocket: Full-Text Searchable Wikipedia on a Raspberry Pi. In *Proceedings of the 15th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL 2015).* Knoxville, Tennessee, 85–86.

[14] Jimmy Lin, Charles L.A. Clarke, and Gaurav Baruah. 2016. Searching from Mars. *IEEE Internet Computing* 20, 1 (2016), 78–82.

[15] Jimmy Lin and Kareem El Gebaly. 2016. The Future of Big Data Is... JavaScript? *IEEE Internet Computing* 20, 5 (2016), 82–88.

[16] Jimmy Lin, Zhucheng Tu, Michael Rose, and Patrick White. 2016. Prizm: A Wireless Access Point for Proxy-Based Web Lifelogging. In *Proceedings of the First Workshop on Lifelogging Tools and Applications (LTA 2016).* Amsterdam, The Netherlands, 19–25.

[17] David A. Patterson. 2008. The Data Center is the Computer. *Commun. ACM* 52, 1 (2008), 105.

[18] Neil Savage. 2018. Going Serverless. *Commun. ACM* 61, 2 (2018), 15–16.

[19] Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. 2017. Distributed Deep Neural Networks over the Cloud, the Edge and End Devices. In *Proceedings of the 37th IEEE International Conference on Distributed Computing Systems (ICDCS 2017).* Atlanta, Georgia, 328–339.

[20] Zhucheng Tu, Mengping Li, and Jimmy Lin. 2018. Pay-Per-Request Deployment of Neural Network Models Using Serverless Architectures. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations.* New Orleans, Louisiana, 6–10.

[21] Sarah K. Tyler and Jaime Teevan. 2010. Large Scale Query Log Analysis of Re-Finding. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining (WSDM 2010).* New York, New York, 191–200.

[22] Ryen W. White and Eric Horvitz. 2017. Evaluation of the Feasibility of Screening Patients for Early Signs of Lung Carcinoma in Web Search Logs. *JAMA Oncology* 3, 3 (2017), 398–401.