Sky Computing: Opportunities and Challenges

Ion Stoica

Abstract In this chapter, we explore the potential evolution of the cloud computing ecosystem, emphasizing the similarities and difference to the developmental trajectories of telephony, the Internet, and PCs. So far the cloud computing market has been dominated by proprietary interfaces from its early entrants, like Amazon Web Services, Windows Azure, and Google Compute Engine. However, in recent years has been an increasing drive from organizations towards enhanced compatibility so they can manage their workloads across various clouds. Unfortunately, unlike the emergence of universal technology standards in previous tech domains, we argue that a comprehensive compatibility standard for the cloud is neither imminent nor necessary for facilitating workload mobility across clouds and could potentially hinder innovation. Instead, we advocate for the adoption of "intercloud brokers"–systems that optimize workload placement based on various customer criteria, such as price and performance, thereby eliminating the need for customers to navigate through cloud selection. This approach, which we term "Sky Computing," is anticipated to lower entry barriers to cloud usage, spur technical innovation through specialized clouds, facilitate comprehensive integration of computational options, and enhance compliance, security, and resilience via cross-cloud deployments. We hope the opportunities and challenges we describe in this chapter will help trigger a broader collaboration effort between researchers and practitioners to develop Sky Computing, steering the future of cloud computing towards this vision.

Keywords Sky computing · Cloud computing · Distributed Systems

I. Stoica (✉)
University of California, Berkeley, Berkeley, CA 04720, USA
e-mail: istoica@berkeley.edu

V. E. Balas et al. (eds.), *Decision Making and Decision Support in the Information Era*,
Studies in Systems, Decision and Control 534,
https://doi.org/10.1007/978-3-031-62158-1_2

1 Introduction

Cloud computing has emerged as a cost-effective alternative to on-premise computing, initially catering to individual customers and thus, not prioritizing cross-platform compatibility. The early cloud providers opted for proprietary interfaces, which, along with strategies like volume discounts and higher data-transfer fees for egress than ingress, have become part of a broader strategy to lock customers into their platforms, making it challenging to shift workloads between different clouds. We observe that if left unchecked, the cloud ecosystem is likely to persist in this trajectory, focusing on lock-in business strategies centered around proprietary services.

However, we, as part of the research community, envision a different trajectory for the future of cloud computing, one that does not succumb to the seemingly inevitable fate of customer lock-in and proprietary dominance. We advocate for the wider research community to steer cloud computing towards a future that embraces compatibility, where workloads can be effortlessly transitioned between clouds. Such compatibility is imperative to meet the escalating demands for data and operational sovereignty, adhering to government regulations that mandate personal data to be operated on within specific geographical and operational parameters. Moreover, this compatibility is vital for corporate users to avoid being intrinsically bound to a single cloud provider, ensuring they retain commercial leverage and have a safeguard against potential major outages.

Traditional approaches to achieving compatibility in technology ecosystems, such as adopting uniform and comprehensive standards, have repeatedly failed in the realm of cloud computing. This is not due to a lack of effort or insurmountable technical challenges, but rather because dominant clouds have little incentive to adopt such standards, which would potentially undermine their competitive edge. Furthermore, the enormity and evolution of APIs, especially with platforms like Amazon Web Service (AWS) offering over 200 services, make it practically impossible to develop a set of comprehensive standards that encompass all these APIs.

Moreover, we argue that adopting comprehensive standards for cloud services could potentially hinder innovation by confining the ecosystem to a set of interfaces that may not cater to future uses. While all technology ecosystems run the risk of premature or inappropriate standardization, this concern is particularly pertinent to cloud computing. The interfaces through which users interact with the cloud span from low-level orchestration to high-level services, making it unclear which levels are apt for standardization and when they are stable enough to be standardized due to their relatively rapid evolution over time.

Our proposal advocates for an alternative vision to create a more compatible yet still evolvable cloud ecosystem, laying its foundation on the use of "intercloud brokers." Users, through an intercloud broker, specify their job characteristics and constraints, along with their desired optimization metrics (such as price or performance). The intercloud broker then determines the optimal cloud for executing each portion of the job and supervises its execution. This approach alters the fundamental

cloud abstraction, enabling users to interact with a more coherent "Sky of Computing" instead of a collection of individual and deliberately differentiated clouds. It establishes a fine-grained, two-sided market, where users specify their requirements and clouds define their offerings, significantly mitigating lock-in effects. Furthermore, our proposal, dubbed "Sky Computing," imposes no requirements on clouds but leverages the incentives created by this two-sided market to encourage their active participation.

In essence, even dominant clouds might perceive Sky Computing as being in their interest, as its enhanced ease of use—transforming the cloud experience from a complex one, where users need to find the optimal combination of services, regions, and resource configurations, to a much simpler one of submitting high- level workload descriptions to an intercloud service broker—might lead to a rapid expansion of the overall cloud market. Sky Computing, while not a panacea, creates a market dynamic where a large class of workloads becomes sufficiently portable, addressing concerns like compliance with various placement regulations and avoiding reliance on a single cloud provider through a process of co-evolution. It is not merely an implementation of"multi-cloud" as currently envisioned but abstracts away clouds with intercloud brokers, achieving partial compatibility and fostering a competitive market without the rigidity of universal standards that could stifle innovation. We believe that with active involvement from the research community, Sky Computing can reshape the future of cloud computing, creating a more resilient and secure computing infrastructure.

2 Related Work

This proposal touches on a many topics, so we cannot adequately review the entire body of related literature. Instead, within the body of the text we cite specific works that are relevant to our discussion. However, there are two topics that we want to mention here because they are so central to our proposal.

First, there have been several attempts to create more compatibility among clouds. In fact, we are not the first to use the name "Sky Computing" in such a proposal; see the following papers, dating back to 2009, for early uses of this term [1–3]. However, these papers argue for a uniform API to be implemented on top of all clouds. One example is Nimbus [4], a cross-cloud Infrastructure-as- a-Service platform targeting specific workloads such as high-performance computing (HPC). In contrast, our proposal does not seek to impose a uniform API. Instead, we are enabling a two-sided market between the cloud providers and application developers; clouds can decide to adopt interfaces in the compatibility set or not (and these will change over time), based on what is best for their business. The central thesis behind Sky Computing is that once this two-sided market is established, greater compatibility will emerge without it having to be imposed, and that the lack of fully uniform and comprehensive standards will encourage innovation.

Organizations are increasingly stating they are multi-cloud, but in most cases this means that different teams in the same organization run different applications on different clouds: e.g., one team running its data analytics workload on Azure while another team runs its machine learning workload on GCP. In contrast, with Sky Computing a single application would take advantage of services offered by multiple clouds to reduce costs and improve the overall application performance. Another multi-cloud scenario is when the same third-party application runs on multiple clouds; for instance, Databricks, Snowflake, and Confluent making their services available on different clouds. However, these services do not abstract away the clouds: a user must select a cloud (and sometimes a region) before creating an account. In contrast, with Sky Computing, the user leverages the intercloud broker to dynamically optimize the mixture of cloud services on which their applications run in response to a continuously changing cloud market.

3 Sky Architecture

Sky Computing is a daunting endeavour, not only because of the significant research challenges, but also because it involves intermediating between real users and real clouds. To learn what problems we might encounter along the way, we will start with simple prototypes that will allow us to test ideas early in our investigation.

3.1 Architecture

At its core, Sky Computing is cloud computing mediated by an intercloud broker (see Fig. 1). In Sky Computing, rather than users directly engaging with a particular cloud, users send a (distributed) application and its description to an intercloud broker, which selects the clouds on which various parts of the application are run and then manages their execution. Thus, an intercloud broker creates a *two-sided market* between users who offer applications, and clouds that offer services. Many of these services (e.g., Kubernetes [5], Apache Spark [6], Apache Kafka [7]) are offered by multiple clouds, while others are cloud specific (e.g., AWS Inferentia [8], BigQuery [9]). We expect that there will be multiple intercloud brokers, some targeting specific workloads. For instance, our first prototype broker focuses on ML training workloads.

To illustrate how Sky works, consider the example in Fig. 2, a machine learning pipeline consisting of three stages: data processing, training, and model serving. Assume that the input data contains confidential information and that the user's goal is to minimize cost. In our example, we use the Amazon Customer Reviews Dataset [10] and treat it as if it contained personally identifiable information (PII) and thus must be processed securely. To remove sensitive data, we run Opaque [11] on an SGX-enabled instance to filter on a column (i.e., the filtered-out information is assumed sensitive), and output only the review texts and star ratings. For training, we use BERT, a popular

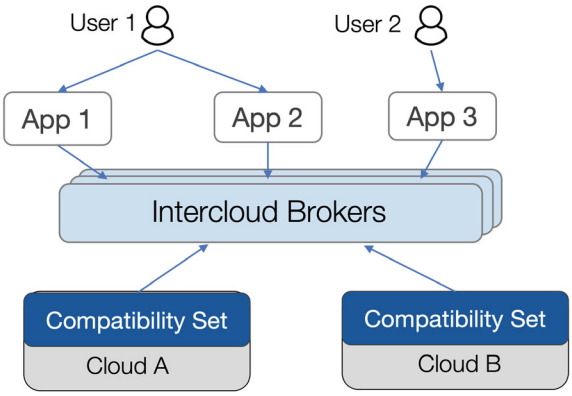

Fig. 1 Sky computing architecture. For different workloads we might have different intercloud brokers

natural language understanding model, on the preprocessed (nonsensitive) data. This model predicts a rating given a review. Since the first stage requires instances based on SGX, today we would be restricted to use Azure Confidential Computing (ACC). In contrast, with Sky we can use ACC only to filter out the PII data, then use GCP for training and AWS for serving to take advantage of their lower cost. So, does Sky help? The answer is "yes". Assuming an 1 GB review dataset, fine-tuning the model for 10 epochs, and serving 1 M queries, sky is *61% cheaper* than executing the same pipeline on Azure, even when factoring in the egress charges to move the data between clouds.

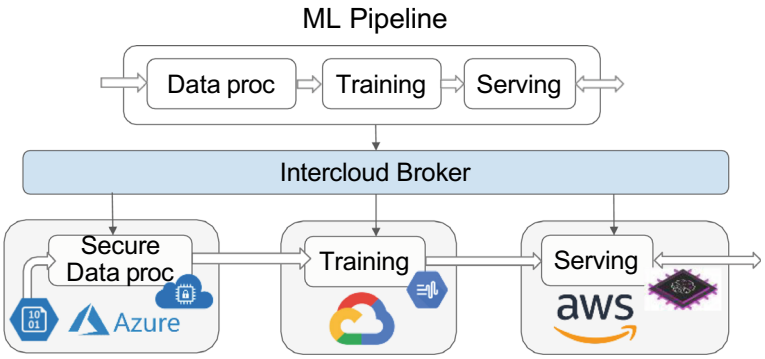

Fig. 2 A simple sky application: a three-stage ML pipeline where each stage runs on a different clou

3.2 *Intercloud Broker*

The intercloud broker is the key component of Sky as it mediates the interaction between users and clouds. An intercloud broker needs to implement the following set of components (see Fig. 3):

- *Service catalog*: This is a list of the service interfaces in the cloud ecosystem, along with the set of clouds that support each service. Each (service, cloud) entry contains instructions on how to instantiate and manage the service on that cloud along with price and performance information.
- *Sky API*: This API allows users to specify their application (where the input data resides, what processing is required, how data flows, etc.) along with their optimization metrics (e.g., cost or delay) and constraints (e.g., process data within the borders of a particular country).
- *Optimizer*: Given the user's job specifications and requirements, the optimizer generates the execution plan (e.g., selecting which clouds, which services, which service configurations, which locations). The execution plan might also involve moving data to a different region in the same cloud or another cloud.
- *Data orchestration*: There are two ways to process data stored in a particular cloud region: run the computation stage in the same region or run the stage in a different region (and possibly cloud) but first move the data to that region. Data orchestration is in charge of moving or replicating the data to the region where the optimizer has decided to run the stage.

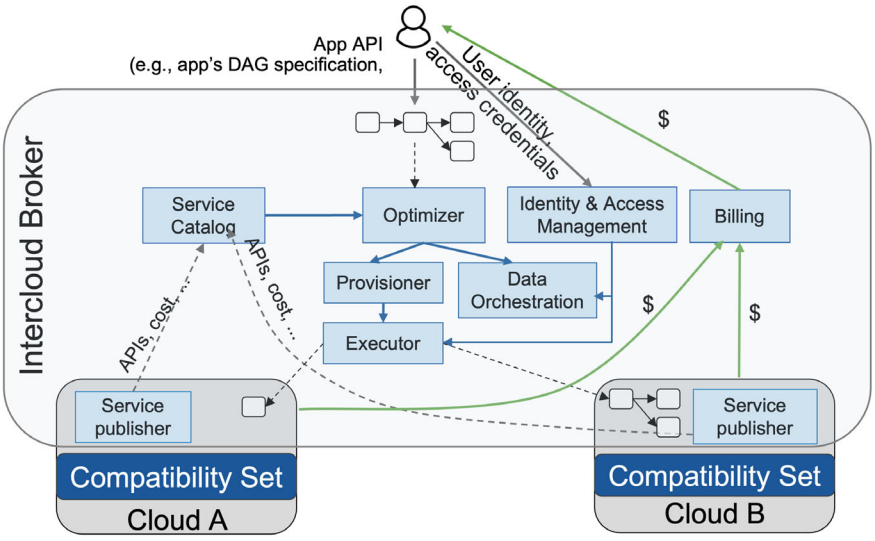

Fig. 3 The intercloud broker architecture

- *Provisioner*: The provisioner is responsible for instantiating and managing services across clouds to execute the physical plan. If the provisioner fails to instantiate the service on the cloud or region (e.g., because the are not enough resources), it can ask the optimizer to generate another physical plan.
- *Executor*: The executor orchestrates the physical computational plan on the resources allocated by the provisioner, detecting and restarting failures and providing some degree of troubleshooting.
- *Billing service (optional)*: We expect several different models will coexist. In some cases, each cloud will bill the users directly for the portion of the job that they ran. In other cases, an intercloud broker could offer its own billing service, in which case it would have contracts with each cloud provider, pay for the job execution on behalf of the user, and then charge the user an aggregate fee.
- *Identity and access management (IAM)*: This component will vary depending on the nature of the job, but at the very least the user must provide input data to the job, and that will likely involve passing some credentials (e.g., tokens) to the intercloud broker. Sky could leverage some of the existing multi-party identity protocols and services, e.g., Kerberos [12], Globus Auth [13], Okta [14].

3.3 Early Results

As our first application, we are planning to focus on supporting training and hyper-parameter tuning for ML. We will focus on reducing the cost, and we will target ML researchers. There are several reasons for these choices. First, while narrow, this use case drives the biggest expenses for many research groups (e.g., it accounts for over 80% of RISE Lab cloud usage). Second, we are familiar with these users. Third, these users already have access to a variety of computation platforms such as on-prem clusters, and public clouds. Fourth, due to the simplicity of this application, we believe that we can experiment and learn quickly. This use case will allow us to start with a relatively simple intercloud broker whose components do not fully implement the functionalities described above. For example, we can start with handling only batch jobs that are simply described as directed acyclic graphs (DAGs), where nodes are training tasks and edges data dependencies. In the simplest instantiation, the IAM needs only limit access to the submitted job, so sophisticated access controls are unnecessary. Furthermore, the initial version of the intercloud broker may not implement billing, and instead require users to have accounts with every cloud they want to use. Other possible applications are distributed analytics [15], cloud robotics [16] and autonomous driving [17].

4 Research Challenges in Building Sky

With few exceptions, most distributed systems assume a set of homogeneous nodes in the same administrative domain that can freely communicate with each other. However, with Sky, we are forced to consider a more complex model. Instead of a flat hierarchy, we have a multi-level hierarchy where the nodes of the same system can belong to different clouds, regions, and datacenters, and where clouds represent different administrative domains. This leads to far more complex connectivity, failure, and trust models, which will require new solutions. Next, we list a few of the research challenges in building Sky, challenges that are driven in part by this highly heterogeneous distributed computation model.

4.1 Developing Sky Applications

Sky programming systems must bridge two APIs: a new Sky API that allows developers to describe their computing desires to brokers, and the APIs in the service catalog of the extant services in the compatibility set. There is already a compatibility set of functionality based on common APIs supported by two clouds or more. This set ranges from storage to VMs to popular higher-level services. For the higher levels, initial compatibility can be assessed via codebase signatures (e.g., the version number of a popular open source package such as Spark or Kafka), via widely adopted standards (e.g., SQL:2008 compatibility) or via careful checking of constrained APIs (e.g., for block storage).

DAGs: An initial broker API for batch jobs could be inspired by workflow systems like Airflow [18]. In this case, the API could be a directed acyclic graph (DAG) where nodes represent (large) computation tasks and edges represent data and ordering dependencies between tasks. A task can specify the cloud services from the service catalog it uses (e.g., Apache Spark v3.2, BigQuery), resource requirements (e.g., number of TPUs or GPUs), and placement constraints (e.g., cloud, country, region). Furthermore, the API allows users to specify their preferences (e.g., use TPUs before GPUs, or use an on-prem cluster before public clouds). We have already implemented an early version of this, and this version is already proving valuable informing us on Sky issues, including functionality, trust, and economics.

Higher-Level Abstractions: The Sky is a new level of abstraction, and ideally this should make things less complex for programmers. Previous high-level programming environments for distributed systems (e.g., [19–21]) have demonstrated potential for declarative programming approaches. This early work often assumed single trust domains and did not offer programmers control over the generated code. There is also narrower-scoped prior work on federated query processing for declarative data processing languages like SQL [22–24] that expose economic costs to varying

degrees. One possibility to address these challenges would be to build out a programming stack that captures a wide range of programmer needs. Following the model of stacks like LLVM [25] and Halide [26] one could pursue a layered approach of Intermediate Representation (IR) languages—each layer having its own syntax open to developers. Developers who want more control over performance/cost tradeoffs should have the ability to author/modify code at lower-level IRs when needed.

Intelligent Assistants: The Sky vision presumes that programmers can navigate the complexity of cloud service APIs. This is not easy: many services have hundreds of functions, obscure documentation, and oft-changing semantics. To navigate this complexity, one could develop *Interactive Program Synthesis* tools that leverage large neural language models such as GPT-3 and Cloude 2 to synthesize code using these APIs. Several existing solutions synthesize code directly from natural language descriptions. While very general, these solutions are unfortunately prone to syntax and semantics errors [27]. Instead of directly synthesizing code, one possibility would be to mine existing available code (e.g., from GitHub) to retrieve idiomatic code snippets solving a given programming task. By composing these snippets, we expect the synthesized code will be meaningful and correct.

4.2 Optimizing Sky Applications

The optimizer is a key component of the intercloud broker and optimizing for sky applications presents several unique challenges. First, the optimization space is combinatorially large, continuously changing, and stateful. For example, imagine a user that submits a training job in the evening to be completed by 9am the next morning. Then, the optimizer is faced with a wide array of *conditional* choices (e.g., use an on-prem cluster if available; if not search for a region with enough spot instances in the same cloud where input is stored; if not, use on-demand instances etc.), must revise its decision as the environment changes (e.g., changes in resource availability and pricing), and finally future actions and costs depend on prior decisions (e.g., whether data was moved to a new region based on the previous pricing). Second, the constants in the optimization problems may not be known. For example, resource availability and pricing, as well as workload performance characteristics and even exact requirements may not be known in advance.

To navigate this complex and dynamic search space in the absence of complete information, researchers will need to develop cost models possibly combining analytical methods with neural network-based approaches similarly to previous work [28, 29]. Ideally, these cost models should be *composable*: estimating the runtime cost of the entire application by summing of the cost for invoking individual tasks on different clouds, rather than profiling all possible executions and comparing their costs. Finally, one way to address the inherent sequential decision making nature of the underlying optimization problem is to combine reinforcement learning (RL) techniques with more traditional optimization methods [30–33].

4.3 Testing and Debugging Sky Applications

Testing, logging, and debugging of Sky distributed applications will add another layer of complexity. One way to reduce such complexities is by building automated testing and debugging tools which will enable developers to detect, localize, and repair bugs. Historical commit data on bug fixes and corresponding execution traces could enable an AI model to help developers understand the cause of a bug.

Despite developing modern automated testing tools for Sky applications, bugs could still escape to production systems. Logging and diagnosing such bugs are often notoriously hard. To address this challenge, one possibility would be to develop AI-based debugging tools to help developers (1) corroborate errors from multiple APIs, (2) localize errors across multiple APIs, (3) understand root causes across multiple layers of abstractions, and (4) handle non-deterministic executions. Furthermore, can leverage language models like GPT-4 and Cloude-2 to suggest logging instructions in the application which will collect necessary debugging information for commonly occurring bugs or exceptions.

4.4 Securing Sky Applications

To secure Sky applications will require designing a security architecture for Sky from the grounds up. The guiding principle here is *least privilege*: each component of Sky should receive only the privileges that it needs to run the relevant task for the user. In particular, the broker should be able to optimize, setup and orchestrate the computation, but it should not receive the credentials for all the users; only a task that directly interfaces with resources such as setting up a VM or accessing data should receive the credentials it needs. This design ensures that if an attacker steals the broker state, the state does not contain all the user's credentials, and if an attacker breaks into a module, the damage it can cause is confined.

Sky should be also compliant with modern data protection and privacy regulations [34, 35] such as ensuring that the user remains in control of their data and what computations will be run on it at all times and that the data can be deleted or user accounts forgotten.

4.5 Data Gravity

Data gravity refers to users running their applications in the cloud where the data is. This is an often cited blocker for multi-cloud applications [36]. Data gravity stems from high transfer latency and egress costs imposed by some clouds when moving data out (up to $0.19 per GB of data moved). We need to develop tools to alleviate this challenge by dramatically reducing the transfer times and the costs. Techniques to

reduce latency of network transfers include overlay routing (e.g., instead of moving data directly from AWS West to Azure East, move data first to AWS East, and then to Azure East), multiple instances and parallel TCP connections to reduce transfer time. Techniques to reduce egress costs include compression to trade egress for CPU cost, selective use of cheaper hot-potato network tiers [37] and private network peering such as AWS Direct Connect [38]. Such a tool will dynamically optimize across these dimensions while taking into account the current network conditions. We have already done simple experiments and the potential of improvements over the state-of-the-art are significant: using a subset of these techniques one can reduce the inter- and intra-cloud transfer times by up to 3–5 × compared to similar tools provided by existing clouds [39, 40]. Combining compression and dynamic network tiering can reduce egress costs by 40–92%.

Furthermore, many more optimizations are possible for structured data where the access is done at the record level. In those cases one can use copy-on-access to minimize data transfer cost at the expense of access latency, various caching techniques (e.g., write through vs write back), as well as prefetching. We expect the optimal solution will depend heavily on the workload (e.g., read heavy vs write heavy), record granularity, as well as the overheads of access control and authentication.

Finally, we make two points. First, even with today's egress costs, these costs are not a show stopper for some workloads that are compute intensive. As a data point, in the ML example in Fig. 2, the egress costs are less than 1% of the compute cost. Second, we expect that some clouds will establish reciprocal peering arrangements, similarly to the free peering between ISPs in the Internet. Some cloud providers are going even further by eliminating the egress costs all together (e.g., Cloudflare's R2 [41]).

5 Summary

The proprietary interfaces of existing cloud providers have paved a path at odds with the desire of more and more organizations to leverage multiple clouds to improve their applications' security, availability, cost efficiency, and avoid lock- in. In this chapter, we propose Sky Computing that creates a tow sided market between the application demands, on one side, and services provided by clouds, on the other side. We believe that Sky Computing will lead to a more compatible and evolvable cloud ecosystem, reducing lock-in effects and potentially expanding the overall cloud market by transforming user experience and achieving partial compatibility through a co-evolution of cloud offerings and workloads. This approach, while technically within reach, necessitates the active involvement of the research community to reshape the future of cloud computing and explore its potential in creating a more resilient and secure computing infrastructure.

References

1. Fortes, J.A.B.: Sky computing: When multiple clouds become one. In: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, pp. 4–4 (2010)
2. Monteiro, A., Pinto, J.S., Teixeira, C.J.V., Batista, T.: Sky computing: exploring the aggregated cloud resources-part i. In: Conference: Information Systems and Technologies (CISTI) (2021)
3. Matsunaga, A., Fortes, J., Keahey, K., Tsugawa. M.: Sky computing. IEEE Internet Comput. **13**(05), 43–51 (2009)
4. Nimbus: Cloud Computing for Science. https://www.anl.gov/mcs/nimbus-cloud-computing-for-science
5. Kubernetes. https://github.com/kubernetes/kubernetes
6. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, pp. 2–2. USENIX Association (2012)
7. Apache Kafka. https://kafka.apache.org/
8. AWS Inferentia. https://aws.amazon.com/machine-learning/inferentia/
9. BigQuery. https://cloud.google.com/bigquery/docs/introduction
10. Amazon Customer Reviews Dataset. https://s3.amazonaws.com/amazon-reviews-pds/readme.html
11. Zheng, W., Dave, A., Beekman, J.G., Popa, R.A., Gonzalez, J.E., Stoica, I.: Opaque: an oblivious and encrypted distributed analytics platform. In: Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation, NSDI'17, pp. 283–298, USA (2017). (USENIX Association)
12. Kerberos: The Network Authentication Protocol. https://web.mit.edu/kerberos/
13. Globus Auth Specification. https://docs.globus.org/api/auth/specification/
14. Okta. https://www.okta.com/
15. Pu, Q., Ananthanarayanan, G., Bod´ık, P., Kandula, S., Akella, A., Bahl, V., Stoica, I.: Low latency geo-distributed data analytics. In: ACM SIGCOMM (Aug 2015)
16. Chen, K., Liang, Y., Jha, N., Ichnowski, J., Danielczuk, M., Gonzalez, J.E., Kubiatowicz, J.D., Goldberg, K.: Fogros: an adaptive framework for automating fog robotics deployment. In: 2021 IEEE 17th International Conference on Automation Science and Engineering (CASE), pp. 2035–2042 (2021)
17. Liu, S., Tang, J., Wang, C., Wang, Q., Gaudiot. J.-L.: Implementing a cloud platform for autonomous driving (2017)
18. Apache Airflow. https://airflow.apache.org/
19. Alvaro, P., et al.: Consistency analysis in bloom: a CALM and collected approach. In: CIDR, pp. 249–260 (2011)
20. Metadata Partners, LLC. Datomic: Technical overview (2012). https://web.archive.org/web/20120324023546. http://datomic.com/docs/datomic-whitepaper.pdf. Accessed 13 Dec 2020
21. Granger, C.: Against the current: What we learned from eve. In: Future of Coding LIVE Conference (2018). https://futureofcoding.org/notes/live/2018
22. Stonebraker, M., Devine, R., Kornacker, M., Litwin, W., Pfeffer, A., Sah, A., Staelin, C: An economic paradigm for query processing and data migration in mariposa. In: Proceedings of 3rd International Conference on Parallel and Distributed Information Systems, pp. 58–67. IEEE (1994)
23. Balazinska, M., Balakrishnan, H., Stonebraker, M.: Contract-based load management in federated distributed systems. In: NSDI, vol. 4, pp. 15–15 (2004)
24. Sethi, R., Traverso, M., Sundstrom, D., Phillips, D., Xie, W., Sun, Y., Yegitbasi, N., Jin, H., Hwang, E., Shingte, N., et al.: Presto: Sql on everything. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE), pp. 1802–1813. IEEE (2019)
25. Lattner, C., Adve, V.: Llvm: a compilation framework for lifelong program analysis & transformation. In: International Symposium on Code Generation and Optimization, 2004. CGO 2004, pp. 75–86. IEEE (2004)

26. Ragan-Kelley, J., Adams, A., Paris, S., Levoy, M., Amarasinghe, S., Durand, F.: Decoupling algorithms from schedules for easy optimization of image processing pipelines. ACM Trans. Graph. **31**(4), 32:1–32:12 (2012)
27. Smith, T.: Why OpenAI's Codex Won't Replace Coders (Sept 2021)
28. Zheng, L., Jia, C., Sun, M., Wu, Z., Yu, C.H., Haj-Ali, A., Wang, Y., Yang, J., Zhuo, D., Sen, K., Gonzalez, J.E., Stoica, I.: Ansor: generating high-performance tensor programs for deep learning. In: 14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4–6, 2020, pp. 863–879 (2020). USENIX Association 2020
29. Zheng, L., Li, Z., Zhang, H., Zhuang, Y., Chen, Z., Huang, Y., Wang, Y., Xu, Y., Zhuo, D., Gonzalez, J.E., Stoica, I.: Alpa: automating inter- and intra-operator parallelism for distributed deep learning (2022). CoRR, abs/2201.12023
30. Liang, E., Zhu, H., Jin, X., Stoica, I.: Neural packet classification. In: Proceedings of the ACM Special Interest Group on Data Communication, SIG-COMM'19, pp. 256–269. New York, NY, USA (2019). Association for Computing Machinery
31. Jain, P., Huda, S., Maas, M., Gonzalez, J.E., Stoical, I., Mirhoseini, A.: Learning to design accurate deep learning accelerators with in accurate multipliers. In: 2022 Design, Automation and Test in Europe Conference & Exhibition (DATE), pp. 184–189 (2022)
32. Krishnan, S., Yang, Z., Goldberg, K., Hellerstein, J., Stoica, I.: Learning to optimize join queries with deep reinforcement learning (2018)
33. Yang, Z., Chiang, W.-L., Luan, S., Mittal, G., Luo, M., Stoica, I.: Balsa: learning a query optimizer without expert demonstrations. In: Proceedings of the 2022 International Conference on Management of Data, SIG-MOD/PODS'22, pp. 931–944. New York, NY, USA (2022). Association for Computing Machinery
34. General data protection regulation (GDPR). https://gdpr-info.eu/
35. California consumer privacy act (CCPA). https://oag.ca.gov/privacy/ccpa
36. Why Data Gravity Is the Single Biggest Challenge for Digital Transformation. https://www.digitalrealty.com/blog/why-data-gravity-is-the-single-biggest-challenge-for-digital-transformation
37. Network Service Tiers overview. https://cloud.google.com/networktiers/docs/overview
38. AWS Direct Connect. https://aws.amazon.com/directconnect/
39. Google Cloud, Storage Transfer Service. https://cloud.google.com/storage-transfer-service
40. AWS DataSync. https://aws.amazon.com/datasync/
41. Announcing cloudflare r2 storage: Rapid and reliable object storage, minus the egress fees. https://blog.cloudflare.com/introducing-r2-object-storage/. Accessed 29 Jan 2022