Serverless End Game: Disaggregation enabling Transparency

Pedro Garcia Lopez
Universitat Rovira i Virgili, Spain
pedro.garcia@urv.cat

Aleksander Slominski
IBM Watson Research, USA
aslom@us.ibm.com

Simon Shillaker
Imperial College London, UK
s.shillaker17@imperial.ac.uk

Michael Behrendt
IBM Deutschland Research &
Development GmbH, DE
michaelbehrendt@de.ibm.com

Bernard Metzler
IBM Research Zurich, CH
bmt@zurich.ibm.com

ABSTRACT

For many years, the distributed systems community has struggled to smooth the transition from local to remote computing. Transparency means concealing the complexities of distributed programming like remote locations, failures or scaling. For us, full transparency implies that we can compile, debug and run unmodified single-machine code over effectively unlimited compute, storage, and memory resources.

We elaborate in this article why resource disaggregation in serverless computing is the definitive catalyst to enable full transparency in the Cloud. We demonstrate with two experiments that we can achieve transparency today over disaggregated serverless resources and obtain comparable performance to local executions. We also show that locality cannot be neglected for many problems and we present five open research challenges: granular middleware and locality, memory disaggregation, virtualization, elastic programming models, and optimized deployment.

If full transparency is possible, who needs explicit use of middleware if you can treat remote entities as local ones? Can we close the curtains of distributed systems complexity for the majority of users?

CCS CONCEPTS

• **Computing methodologies → Parallel computing methodologies**; **Distributed computing methodologies**.

KEYWORDS

Transparency, Disaggregation, Serverless

1 INTRODUCTION

Transparency is an archetypal challenge in distributed systems that has not yet been adequately solved. Transparency implies the concealment from the user and the application programmer of the complexities of distributed systems. Colouris et al. [12] define eight forms of transparency: access, location, concurrency, replication, failure, mobility, performance, and scalability.

But, despite all previous efforts, the problem is still open as seen in recent literature. For example, as stated in [21]: "Our proposal in this paper was motivated by a professor of computer graphics at UC Berkeley asking us: Why is there no cloud button?" He outlined how his students simply wish they could easily push a button and have their code (existing, optimized, single-machine code) running on the cloud.

Waldo et al. [44] explain that the goal of merging the programming and computational models of local and remote computing is not new. They state that "around every ten years a furious bout of language and protocol design takes place and a new distributed computing paradigm is announced". They mention messages in the 70s, RPCs in the 80s, and objects in the 90s.

In every iteration, a new wave of software modernization is generated, and applications are ported to the newest and hot paradigm. Waldo et al. claim that all these iterations may be evolutionary stages to unify both local and distributed computing. But they are pessimistic, and they believe that this will not be possible because of latency, memory access, concurrency and partial failure.

This visionary paper even considers that in the future hardware improvements could make the difference in latency irrelevant, and that differences between local and remote memory could be masked. But they still claim that concurrency and partial failures preclude the unification of local and remote computing. Unlike an OS, they are telling us that a distributed system has no single point of resource allocation, synchronization, or failure.

But, what if novel cloud technologies could make the unification of local and remote paradigms possible? Are we close to the end of the cycles of software modernization? Can we just compile to the Serverless SuperComputer [43]?

This paper argues that recent reductions in network latency [6, 36] are boosting resource disaggregation in the Cloud, which is the definitive catalyst to achieve transparency. Even if existing Cloud services are still in the millisecond range (100ms Lambda overhead, 10ms in Kafka, 5-20ms in S3), disaggregation has already fueled the creation of serverless computing services like Function as a Service, Cloud Object Storage, and messaging. If we can go down to µs RPCs [22, 24], novel opportunities for transparency will emerge [6, 25].

The Serverless End Game (enabling transparency) will arrive when all computing resources can be offered in a disaggregated way. In this paper, we analyze the current research challenges that need to be addressed in order to achieve this ambitious goal.

SESAME '24, April 22, 2024, Athens, Greece

ACM ISBN 979-8-4007-0545-8/24/04…$15.00
https://doi.org/10.1145/3642977.3652094

2 DDC PATH TO TRANSPARENCY

The DDC path is probably the more direct but also the more shocking for the distributed systems community. In line with recent industrial trends on Disaggregated Data centers (DDC) [15], it implies a distributed OS transparently leveraging disaggregated hardware resources like processing, memory or storage.

A canonical example is LegoOS: A disseminated, distributed OS for hardware resource disaggregation [38]. LegoOS exposes a distributed set of virtual nodes (vNode) to users. Each vNode is like a virtual machine managing its own disaggregated processing, memory and storage resources. LegoOS achieves transparency and backwards compatibility by supporting the Linux system call interface and Linux ABIs (Application Binary Interface), so that existing unmodified Linux applications can run on top of it. Even distributed applications that run on Linux can seamlessly run on a LegoOS cluster by running on a set of vNodes. For example, LegoOS shows how two unmodified applications can be run in a distributed way: Phoenix (a single-node multi-threaded implementation of MapReduce) and TensorFlow.

Another relevant work is Arrakis: The Operating System is the Control Plane [34]. Arrakis comes from previous efforts aimed at optimizing the kernel code paths to improve data transfer and latency in the OS. In Arrakis, applications have direct access to virtualized I/O devices, which allows most I/O operations to bypass the kernel entirely without compromising process isolation. Arrakis virtualized control plane approach allows storage solutions to be integrated with applications, even allowing the development of higher level abstractions like persistent data structures. Even more, Arrakis control plane is a first step towards integration with a distributed data center network resource allocator.

If the OS can be extended with unbounded resources in a transparent way, distribution may no longer be needed for many applications – single-node parallel programming is sufficient. This is completely in line with the following assessment from the COST paper [29]: "You can have a second computer once you've shown you know how to use the first one". This paper presents a critique of the current research in distributed systems, and even suggests that "there are numerous examples of scalable algorithms and computational models; one only needs to look back to the parallel computing research of decades past".

COST stands in that paper for the "Configuration that Outperforms a Single Thread". They mainly compare optimized single-threaded versions of graph algorithms, with their equivalents in distributed frameworks like Spark, Naiad, GraphX, Giraph or GraphLab. For example, Naiad has a COST of 16 cores for executing PageRank on the twitterrv graph, which means that Naiad needs 16 cores to outperform a single-threaded version of the same algorithm in one machine.

An important reflection from this paper is that the overheads of distributed frameworks (coordination, serialization) can be extremely high just in order to justify scalability. But the COST paper is not proposing a solution to the scalability problem, since it is obvious that a single machine cannot scale enough for many algorithms.

But, what happens if we combine the COST idea with the DDC research? This is precisely what Gao et al.[15] validated in a simple experiment comparing a COST version with a COST-DDC one that relies on disaggregated memory (Infiniswap [17]). They demonstrate in this paper that the same code can overcome the memory limits thanks to disaggregation and still obtain good performance results.

DDC is openly challenging the so-called server-centric approach of development for the data center. DDC advocates that the monolithic server model where the server is the unit of deployment, operation, and failure is becoming obsolete. However, current mature multi-tenant Cloud technologies are built on top of server-centric models which are still difficult to challenge by DDC proposals.

3 SERVER-CENTRIC PATH TO TRANSPARENCY

Recent proposals are intercepting language libraries in order to access remote Cloud resources in a transparent way. For example, Crucial [5] implements a Serverless Scheduler for the Java Concurrency library. Crucial can run Java threads in Serverless functions transparently, and it also provides synchronization primitives and consistent mutable state data structures over a disaggregated in-memory computing layer. Crucial does not provide flexible memory scaling or storage transparency, and it is limited to Java applications using that library.

In [4], authors intercept Python multiprocessing library to transparently execute Python applications at scale over Cloud serverless resources. This paper demonstrates that transparency is feasible for many unmodified existing applications. However, they show that for read-write memory intensive applications, transparency may involve huge penalties.

Another example of language level transparency is Fiber [45]. Fiber implements an alternative Python multiprocessing library that works over a scalable Kubernertes cluster. Fiber supports many Python multiprocessing abstractions like Process, Pool, Queue, Pipe and also remote memory in Manager objects. It demonstrates transparency executing unmodified Python applications from the OpenAI Baselines machine learning project. But Fiber does not support transparent disaggregated storage and memory, and it is limited to Python applications using that library.

The Fabric for Deep Learning (FfDL) [20] system moves existing Deep Learning frameworks like PyTorch or TensorFlow to the Cloud on top of cluster technologies like Kubernetes. [20] transparently provides dependability thanks to checkpointing, intercepting storage flows (file system) using optimized storage drivers to cloud object storage, and supporting locality with a gang scheduling algorithm that schedules all components of a job as a group. However, they observed that scaling was so framework dependent that they could not achieve full scaling transparency.

Another example of transparency in a serverless context is Faasm [40]. Faasm exposes a specialised system interface which includes some POSIX syscalls, serverless-specific tasks, and frameworks such as OpenMP and MPI. Faasm transparently intercepts calls to this interface to automatically distribute unmodified applications, and execute existing HPC applications over serverless compute resources.

Faasm allows colocated functions to share pages of memory and synchronises these pages across hosts to provide distributed state.

However, this is done through a custom API where the user must have knowledge of the underlying system, hence breaking full transparency. Furthermore, when functions are widely distributed, this approach exhibits performance similar to traditional distributed shared memory (DSM), which has proven to be poor without hardware support [11, 32].

4 LIMITS OF DISAGGREGATION AND TRANSPARENCY

Current data center networks already enable disk storage disaggregation [3], where reads from local disk are comparable (10ms) to reads from the network. In contrast, creating a thread in Linux takes about 10 µs, still far from the 15ms/100ms (warm/cold) achieved today in FaaS settings. With that, compute disaggregation is already feasible when job time renders these delays negligible.

Advances in datacenter networking and NVMs have reduced access to networked storage to 1 µs, however this is still an order of magnitude slower than local memory accesses which are in the nanosecond range [6] (100ns), and local cache accesses in the 4ns-30ns range. This means that local memory cannot be neglected, and should be smartly leveraged by memory disaggregation efforts [27]. Existing efforts in memory disaggregation [13, 17, 23, 33] strive to play in the µs range, which can be a limiting factor.

This is directly related to locality and affinity requirements for many stateful applications. The systems community is starting to acknowledge that stateful services need a different programming model and resource management than the stateless ones [18, 25]. Stateful services have very different requirements of coordination, consistency, scalability and fault tolerance, and they need to be addressed differently. Stateful services show the limits of disaggregation versus locality, since in some scenarios locality still matters.

For now, locality still plays a key role in stateful distributed applications. For example: (i) where huge data movements still are a penalty and memory-locality can be still useful to avoid data serialization costs; (ii) where specialized hardware like GPUs must be used [20]; in (iii) some iterative machine-learning algorithms [19]; in (iv) simulators, interactive agents or actors[35].

Finally, another important limitation is scaling transparency, which means that applications can expand in scale without changes to the system structure or the application algorithms. If the local programming model was designed to use a fixed amount of resources, there is no magic way of transparently achieving scalability, not to mention elasticity. Workloads that do not need elasticity, such as enterprise batch jobs or scientific simulations, can use disaggregated resources the same way as local as they do not need scalability. However, for more user driven and interactive services, such as internal enterprise web applications, simple porting of the executables (sometimes referred as “lift-and-shift“) is rarely enough. The unchanged code is not able to take advantage of the elasticity of disaggregated resources and it is expensive to run code that is not used.

5 EXPERIMENTS

5.1 Compute disaggregation

To evaluate the feasibility of compute disaggregation with state of the art cloud technologies, we will compare a compute-intensive algorithm running in local threads in a VM compared to the same algorithm running over serverless functions. We also provide code transparency, since we execute the same code in both cases. To achieve this transparency, we rely on a Java Serverless Executor [5] that can execute Java threads over remote Lambda functions. In this case, all state is passed as parameters to the functions/threads, and functions are in warm state, like VMs which are already provisioned.

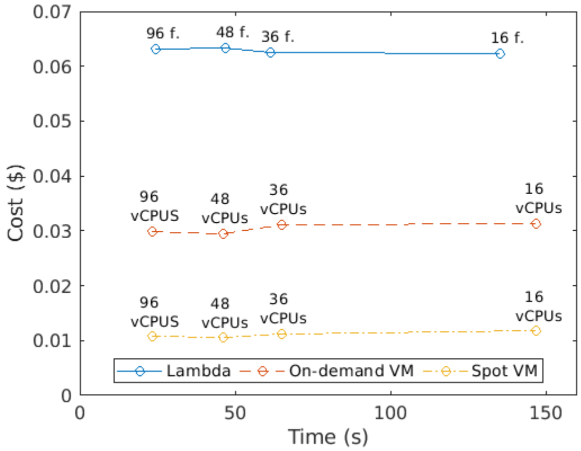

Figure 1: Monte Carlo simulation in VMs versus Amazon Lambda Functions

This experiment runs a Monte Carlo simulation to estimate the value of π. At each iteration, the algorithm checks if a random point in a 2D square space lies inside the inscribed quadrant. We run 48 billion iterations of the algorithm. For AWS Lambda, the iterations are evenly distributed to 16, 36, 48 or 96 functions with 1792 MB of memory.[1] For virtual machines, we run a parallel version of the simulation in different instance sizes: c5.4xlarge (16 vCPUs), c5.9xlarge (36 vCPUs), c5.12xlarge (48 vCPUs), c5.24xlarge (96 vCPUs). The algorithm is implemented in Java.

As we can see in Figure 1, the major difference now is cost: for an equivalent execution, disaggregated functions cost 2x more compared to on-demand VMs, and 6x more compared to Spot instances. Surprisingly, computation time is equivalent in the local and remote version using Lambdas. Even considering all the network communication overheads, container management and remote execution, the results for disaggregated computations are already competitive in performance in existing clouds. This is of course happening because this experiment is embarrassingly parallel, and the duration of compute tasks is long enough to make milliseconds (15/100ms) overheads negligible.

5.2 Memory disaggregation

The second experiment evaluates the feasibility and costs of both memory and compute disaggregation with existing cloud technologies. In this case, we evaluate a linear algebra algorithm, Matrix Multiplication (GEMM) which is a good use-case for testing parallel processing on large in-memory data structures.

[1] According to AWS documentation, at 1,792MB a function has the equivalent of one full vCPU

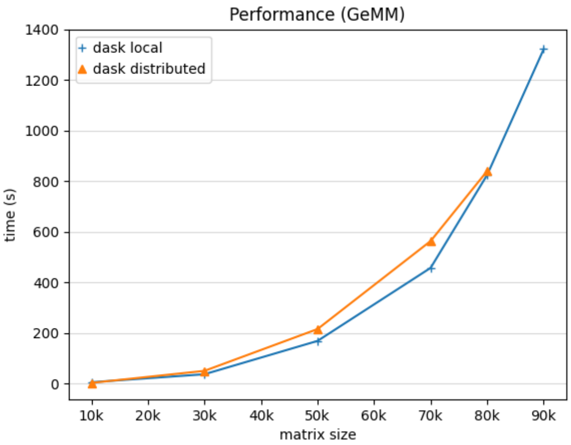

Figure 2: Comparing Vertical vs. Horizontal Scaling: GEMM Matrix Multiplication in Dask Local vs. Distributed

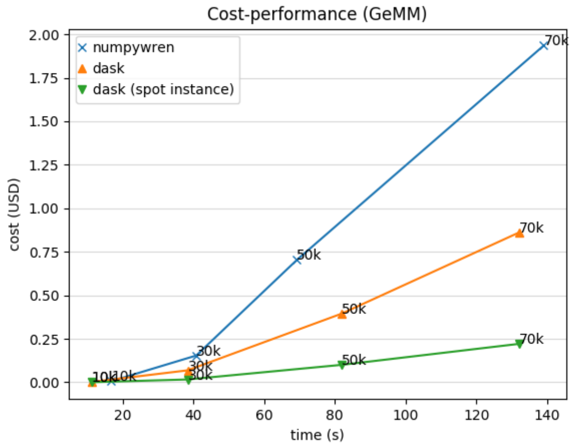

Figure 3: Comparing Horizontal Scaling Options: GEMM Matrix Multiplication in Dask Distributed (Spot Instances and on demand VMs) and numpywren (Lambda) for different matrix sizes

We rely on Python frameworks used by data scientists like NumPy and Dask. Dask transparently enables to run the same code in a single multi-core machine or VM, and in a distributed cluster of nodes. We also compare Dask to a serverless implementation of NumPy called numpywren [39] using serverless functions that access data in disaggregated Cloud Object Storage (Amazon S3).

Our first experiment compares the performance of Matrix Multiplication (GEMM) using Dask in a local VM (1x r5.24xlarge) and in a distributed cluster (6x r5.4xlarge) using the same resources (96 vCPUs, 768 GiB memory, 10Gb network). Figure 2 shows that the local version perform slightly better than the distributed one while costing the same. In this case, locality is avoiding unnecessary data movements and serialization costs, and cluster provisioning. Experiments with 90Kx90K matrices can be executed in the local VM, but not in the equivalent distributed cluster due to resource exhaustion.

Our second experiment compares the cost and performance of Matrix Multiplication (GEMM) using Dask in a distributed cluster (on demand VMs or Spot instances) and using numpywren over Amazon Lambda and Amazon S3. We calculate compute resources in numpywren (vCPUs) as the ratio between the sum of the duration of every Lambda and the wall-clock time of the experiment. In GeMM (70Kx70K) numpywren uses 553.8 vCPUs and in Dask we use equivalent resources: 552 vCPUs (5x c5.24xlarge, 1x c5.18xlarge).

Figure 3 shows that Dask obtains the same performance in VMs and Spot instances, but Spot instances are 4x cheaper than on demand VMs. numpywren obtains good performance numbers for large matrices, obtaining equivalent performance results for an equivalent Dask cluster in running time. numpywren also shows automatic scaling for any size, whereas the Dask cluster must always be provisioned in advance with the desired amount of resources. Finally, numpywren is much more expensive than the Dask cluster using Spot instances (14x for 10K, 9x for 30K, 6.9x for 50K, 8.7x for 70K).

We see in these experiments what can be achieved today with existing state-of-the-art Cloud infrastructure. Monetary cost is now the strongest reason for locality in Cloud providers as we see in the pricing models for Lambda, on demand VMs and Spot instances. But even if elastic disaggregated resources are now more expensive, some large scale compute intensive problems like linear algebra are now already competitive in compute time and scalability. Further improvements in cloud management control planes and locality-aware placement could reduce costs for elastic resources.

6 CHALLENGES AHEAD

Let us review the major challenges to enable transparency for many applications:

- **Granular middleware and locality**: In line with granular computing [6, 25], we require microsecond latencies in existing middleware (compute, storage, memory, communication). In particular, there is a need to handle extremely short instantiation and execution times and more lightweight container technologies. We also require microsecond latencies in disaggregated storage and memory, messaging and collective communication.
  Granular applications are amenable to fine-grained elastic scaling, but this will not provide adequate performance without data locality. Locality and fine-grained resource management may also reduce the current cost of disaggregated resources. Locality is also needed to scale stateful services with different requirements of coordination, concurrency, consistency, distribution, scalability and fault tolerance. A recent paper [26] shows how granular computing and computing could be combined to achieve millisecond latencies in large flash bursts benefiting from locality. This clearly connects with bursting group behaviours advocated before for Serverless Clusters [31]. We foresee that next-generation

container technologies may enable inter-container communication and provide affinity services for grouping related entities.

- **Memory disaggregation and Computational memory**: Disaggregated memory is still an open challenge and there is no available Cloud offering in this line. Many cluster technologies like Apache Spark, Dask, or Apache Ray rely on coupled and difficult-to-scale in-memory storage. Fast disaggregated memory and storage services [13, 23] can facilitate the elasticity of many cluster technologies [42].
  An important problem here is that disaggregated memory services cannot ignore the memory available in existing server-centric nodes in most Cloud providers. One option is to combine both local and remote memory resources efficiently [27]. Another potential solution here is the recent line on computational memory [37] and in-memory computing devices.
  Fortunately, recent advances are making memory disaggregation a feasible problem in the short term [2] [16]. Further advances in optical communications, as new protocols like CXL will clearly accelerate memory access. Even today, fast networks and non volatile memories (NVM) can be used in supercomputers and data centers with very low latencies.
- **Virtualization** Accessing disaggregated resources in a transparent manner requires a form of lightweight, flexible virtualization that does not currently exist. This virtualization must intercept computation and memory management to provide access to disaggregated resources, and must do so with native-like performance and no input from the programmer. Current serverless platforms use Linux containers and VMs for virtualization [1], which have proven to be too heavyweight for fine-grained scaling, and inappropriate for stateful applications [18, 25, 40]. Software-based virtualization is a more lightweight alternative that is seeing adoption in the serverless context [8, 40], and as a replacement for Docker [30], but is not yet mature enough to transparently support non-trivial existing applications.
  A clear alternative here is to leverage scale-up computing alternatives [14] to pack same tenant code in large containers and VMs. This is a trend we see in serverless settings [7] [41] that also connects with the idea of serverless clusters [31].
- **Elastic programming models and developer experience:** In some cases, virtualization technologies cannot solve problems like scaling transparency if the code is programmed to use a fixed amount of resources. We then need elastic programming models for local machines that can be used without change when running over Cloud resources. Such elastic models should take care of providing the different transparency types (scaling, failure, replication, location, access) and other aspects of application behavior when it is moved between local and distributed environments. The local executable APIs may need to be expanded to include elastic programming abstractions for processes, memory, and storage.
  To fulfill the vision of disaggregation and transparency it will also be critical to provide tools for developers, enabling them to code both locally and remotely in the same manner with full transparency. Developers will need to be able to use tools to debug, monitor, profile, and if necessary access control planes to optimize their applications for cost and performance.
- **Optimized deployment**: Existing applications are a black-box for the cloud, but the transition will imply a "compile to the Cloud" process. In this case, the Cloud will have access over applications' life cycle and it will be able to optimize their execution performance and cost. This means that they can perform static analysis to predict resource requirements, dependencies and potential for hardware acceleration. Future Cloud orchestration services will explicitly leverage data dependencies and execution requirements for improving workloads and resource management thanks to machine learning techniques [10, 28]. This compile process will also allow advanced debugging mechanisms for Cloud applications.
  Transparency efforts for different types of applications will require customizable control planes for applications. Such customization will be based on advanced observability and fast orchestration mechanisms relying on standard services and protocols. Monitoring and interception of the different resources (compute, storage, memory, network) should be available and even integrated into the data center, enabling coordinated actuators at different levels. This can enable the creation of millions of tiny control planes [9] adapted to the different applications and programming models.

7 CONCLUSIONS

We argue that full transparency will be possible soon in the Cloud thanks to low latency resource disaggregation. We foresee that next generation serverless technologies may overcome the limitations exposed by Waldo et al. [44] more than twenty five years ago.

An important paradox of compute disaggregation is the continuous increase in transistor count involving more cpus per chip. The next frontier for transparency is to efficiently combine scale-up computing, scale-out computing and disaggregated memory in an efficient way. Another important challenge is to devise novel elastic parallel programming models for a single machine that effectively guarantee scaling transparency while dealing with partial failures.

Finally, a big challenge is to identify which applications and workloads are prone to achieve first transparency thanks to disaggregation. In principle, parallel data analytics and machine learning are the clear candidates, but other popular settings like Web, mobile and even multi-user games may also be studied.

If transparency is possible soon: Is this the end of distributed programming for the majority of developers? Can we just rely on parallel programming techniques and be completely oblivious to the underlying distributed infrastructure even for large scale problems? Who needs explicit use of middleware if you can treat remote entities as local ones? And finally, can we close the curtains of distributed systems complexity for the majority of users?

ACKNOWLEDGMENTS

This research has been partially funded by the European Union through the NEARDATA project (101092644) and CLOUDSTARS project (101086248),

REFERENCES

[1] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. 2020. Firecracker: Lightweight Virtualization for Serverless Applications. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*. https://www.usenix.org/conference/nsdi20/presentation/agache

[2] Marcos K Aguilera, Emmanuel Amaro, Nadav Amit, Erika Hunhoff, Anil Yelam, and Gerd Zellweger. 2023. Memory disaggregation: why now and what are the challenges. *ACM SIGOPS Operating Systems Review* 57, 1 (2023), 38–46.

[3] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. 2011. Disk-locality in datacenter computing considered irrelevant.. In *HotOS*, Vol. 13. 12–12.

[4] Aitor Arjona, Gerard Finol, and Pedro García López. 2023. Transparent serverless execution of Python multiprocessing applications. *Future Generation Computer Systems* 140 (2023), 436–449.

[5] Daniel Barcelona-Pons, Marc Sánchez-Artigas, Gerard París, Pierre Sutra, and Pedro García-López. 2019. On the FaaS Track: Building Stateful Distributed Applications with Serverless Architectures. In *Proceedings of the 20th International Middleware Conference*. 41–54.

[6] Luiz Barroso, Mike Marty, David Patterson, and Parthasarathy Ranganathan. 2017. Attack of the killer microseconds. *Commun. ACM* 60, 4 (2017), 48–54.

[7] Rohan Basu Roy, Tirthak Patel, Richmond Liew, Yadu Nand Babuji, Ryan Chard, and Devesh Tiwari. 2023. ProPack: Executing Concurrent Serverless Functions Faster and Cheaper. In *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing*. 211–224.

[8] Sol Boucher, Anuj Kalia, David G Andersen, and Michael Kaminsky. 2018. Putting the "Micro" Back in Microservice. *USENIX Annual Technical Conference (USENIX ATC)* (2018). https://www.usenix.org/conference/atc18/presentation/boucher

[9] Marc Brooker, Tao Chen, and Fan Ping. 2020. Millions of Tiny Databases. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 463–478.

[10] David Buchaca, Joan Marcual, Josep LLuis Berral, and David Carrera. 2020. Sequence-to-sequence models for workload interference prediction on batch processing datacenters. *Future Generation Computer Systems* (2020).

[11] John B Carter, John K Bennett, and Willy Zwaenepoel. 1991. Implementation and Performance of Munin. *SIGOPS Oper. Syst. Rev.* (1991). https://doi.org/10.1145/121133.121159

[12] George Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. 2011. *Distributed Systems: Concepts and Design* (5th ed.). Addison-Wesley Publishing Company, USA.

[13] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. 2014. FaRM: Fast remote memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. 401–414.

[14] Grégory M Essertel, Ruby Y Tahboub, James M Decker, Kevin J Brown, Kunle Olukotun, and Tiark Rompf. 2017. Flare: Native compilation for heterogeneous workloads in Apache Spark. *arXiv preprint arXiv:1703.08219* (2017).

[15] Peter X Gao, Akshay Narayan, Sagar Karandikar, Joao Carreira, Sangjin Han, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. 2016. Network requirements for resource disaggregation. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 249–264.

[16] Donghyun Gouk, Sangwon Lee, Miryeong Kwon, and Myoungsoo Jung. 2022. Direct access,{High-Performance} memory disaggregation with {DirectCXL}. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 287–294.

[17] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G Shin. 2017. Efficient memory disaggregation with infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 649–667.

[18] Joseph M Hellerstein, Jose Faleiro, Joseph E Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. 2018. Serverless computing: One step forward, two steps back. *arXiv preprint arXiv:1812.03651* (2018).

[19] Yuzhen Huang, Xiao Yan, Guanxian Jiang, Tatiana Jin, James Cheng, An Xu, Zhanhao Liu, and Shuo Tu. 2019. Tangram: bridging immutable and mutable abstractions for distributed data analytics. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 191–206.

[20] KR Jayaram, Vinod Muthusamy, Parijat Dube, Vatche Ishakian, Chen Wang, Benjamin Herta, Scott Boag, Diana Arroyo, Asser Tantawi, Archit Verma, et al. 2019. FfDL: A Flexible Multi-tenant Deep Learning Platform. In *Proceedings of the 20th International Middleware Conference*. 82–95.

[21] Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. 2017. Occupy the cloud: Distributed computing for the 99%. In *Proceedings of the 2017 Symposium on Cloud Computing*. 445–451.

[22] Anuj Kalia, Michael Kaminsky, and David Andersen. 2019. Datacenter RPCs can be general and fast. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 1–16.

[23] Ana Klimovic, Yawen Wang, Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, and Christos Kozyrakis. 2018. Pocket: Elastic ephemeral storage for serverless analytics. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 427–444.

[24] Marios Kogias, George Prekas, Adrien Ghosn, Jonas Fietz, and Edouard Bugnion. 2019. R2P2: Making RPCs first-class datacenter citizens. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 863–880.

[25] Collin Lee and John Ousterhout. 2019. Granular Computing. In *Proceedings of the Workshop on Hot Topics in Operating Systems*. 149–154.

[26] Yilong Li, Seo Jin Park, and John Ousterhout. 2021. {MilliSort} and {MilliQuery}:{Large-Scale}{Data-Intensive} Computing in Milliseconds. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 593–611.

[27] Ling Liu, Wenqi Cao, Semih Sahin, Qi Zhang, Juhyun Bae, and Yanzhao Wu. 2019. Memory Disaggregation: Research Problems and Opportunities. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1664–1673.

[28] Jonathan Mace, Peter Bodik, Rodrigo Fonseca, and Madanlal Musuvathi. 2015. Retro: Targeted resource management in multi-tenant distributed systems. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*. 589–603.

[29] Frank McSherry, Michael Isard, and Derek G Murray. 2015. Scalability! But at what COST?. In *15th Workshop on Hot Topics in Operating Systems (HotOS 15)*.

[30] Microsoft Research. [n. d.]. Krustlet. https://deislabs.io/posts/introducing-krustlet/

[31] Ingo Müller, Rodrigo FBP Bruno, Ana Klimovic, Gustavo Alonso, John Wilkes, and Eric Sedlar. 2020. Serverless Clusters: The Missing Piece for Interactive Batch Applications?. In *10th Workshop on Systems for Post-Moore Architectures (SPMA 2020)*.

[32] Jacob Nelson, Brandon Holt, Brandon Myers, Preston Briggs, Luis Ceze, Simon Kahan, and Mark Oskin. 2015. Latency-Tolerant Software Distributed Shared Memory. In *2015 USENIX Annual Technical Conference*. https://www.usenix.org/conference/atc15/technical-session/presentation/nelson

[33] John Ousterhout, Parag Agrawal, David Erickson, Christos Kozyrakis, Jacob Leverich, David Mazières, Subhasish Mitra, Aravind Narayanan, Diego Ongaro, Guru Parulkar, et al. 2011. The case for RAMCloud. *Commun. ACM* 54, 7 (2011), 121–130.

[34] Simon Peter, Jialin Li, Irene Zhang, Dan RK Ports, Doug Woos, Arvind Krishnamurthy, Thomas Anderson, and Timothy Roscoe. 2015. Arrakis: The operating system is the control plane. *ACM Transactions on Computer Systems (TOCS)* 33, 4 (2015), 1–30.

[35] RISELab. [n. d.]. Apache Ray. https://github.com/ray-project/ray.

[36] Stephen M Rumble, Diego Ongaro, Ryan Stutsman, Mendel Rosenblum, and John K Ousterhout. 2011. It's Time for Low Latency.. In *HotOS*, Vol. 13. 11–11.

[37] Abu Sebastian, Manuel Le Gallo, Riduan Khaddam-Aljameh, and Evangelos Eleftheriou. 2020. Memory devices and applications for in-memory computing. *Nature Nanotechnology* (2020), 1–16.

[38] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. LegoOS: A disseminated, distributed OS for hardware resource disaggregation. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 69–87.

[39] Vaishaal Shankar, Karl Krauth, Qifan Pu, Eric Jonas, Shivaram Venkataraman, Ion Stoica, Benjamin Recht, and Jonathan Ragan-Kelley. 2018. Numpywren: Serverless linear algebra. *arXiv preprint arXiv:1810.09679* (2018).

[40] Simon Shillaker and Peter Pietzutch. 2020. FAASM: Lightweight Isolation for Efficient Stateful Serverless Computing. In *2020 USENIX Annual Technical Conference (USENIX ATC 19)*.

[41] Jovan Stojkovic, Tianyin Xu, Hubertus Franke, and Josep Torrellas. 2023. MXFaaS: Resource Sharing in Serverless Environments for Parallelism and Efficiency. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–15.

[42] Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, Ana Klimovic, Adrian Schuepbach, and Bernard Metzler. 2019. Unification of temporary storage in the nodekernel architecture. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 767–782.

[43] Tim Wagner. 2019. The Serverless SuperComputer. https://read.acloud.guru/https-medium-com-timawagner-the-serverless-supercomputer-555e93bbfa08

[44] Jim Waldo, Geoff Wyant, Ann Wollrath, and Sam Kendall. 1996. A note on distributed computing. In *International Workshop on Mobile Object Systems*. Springer, 49–64.

[45] Jiale Zhi, Rui Wang, Jeff Clune, and Kenneth O. Stanley. 2020. Fiber: A Platform for Efficient Development and Distributed Training for Reinforcement Learning and Population-Based Methods. *arXiv preprint arXiv:2003.11164* (2020).