Making data center computations fast, but not so furious

Daniel Porto
INESC-ID/IST – U. Lisboa

João Loff
INESC-ID/IST – U. Lisboa

Rui Duarte
INESC-ID/IST – U. Lisboa

Luis Ceze
University of Washington

Rodrigo Rodrigues
INESC-ID/IST – U. Lisboa

ABSTRACT

We propose an aggressive computational sprinting variant for data center environments. While most of previous work on computational sprinting focuses on maximizing the sprinting process while ensuring non-faulty conditions, we take advantage of the existing replication in data centers to push the system beyond its safety limits. In this paper we outline this vision, we survey existing techniques for achieving it, and we present some design ideas for future work in this area.

KEYWORDS

Computational Sprinting, Overclock, Fault tolerance, Data center.

1 INTRODUCTION

Today's global scale Internet services run in large data center infrastructures and are accessed by millions of users. The sheer scale in which these systems operate is such that the design of the infrastructure underlying data center systems must expect an environment where faults are the norm, and no longer the exception.

A key technique for building systems to provide high availability despite faults is to employ redundancy, often through the use of distributed replication protocols. However, this redundancy has a resource usage cost associated with it. For instance, Google uses 5 replicas for the F1 Advertising Back-end [3], which multiplies the number of servers required to run this service by that factor.

Furthermore, F1 is not an isolated example, since redundancy is present in a large fraction of the systems that are part of the software stack of major companies such as Google or Facebook [22]. Thus, redundantly storing data and performing computations in multiple servers increases the energy cost and uses resources that otherwise could be allocated to serve other types of requests. Nevertheless, this expense is seen as an important insurance, because faults lead to service downtime which, in turn, affects revenue [5].

On an orthogonal direction, energy efficiency is also a concern for data center operators, as it impacts the requirements and consequently the cost of the infrastructure for power delivery. This is a pressing problem both because the power consumption of servers is increasing with the advances in density and number of cores, and because the cost associated with an increase in power capacity can be very high, reaching tens of thousands of USD per additional MegaWatt [10].

Moreover, while over-subscription of data centers' power supply allows for accommodating infrequent correlated spikes in server power consumption, it exposes data centers to the risk of tripping power breakers and causing outages. For instance, Facebook initially had to disable dynamic overclocking (Intel Turbo Boost [12]) in one of its clusters, due to an insufficient power margin, despite the potential benefits in performance of this technology. To safely enable turbo mode while avoiding the risk of outages in high load periods, they designed a power management system to cap energy consumption according to the data center power budget [24].

To lower energy consumption while also maintaining and even improving performance, a recent approach called *computational sprinting* [21] exploits the *thermal capacitance* of materials to activate cores (parallel sprinting) or overclocking the CPU (frequency sprinting), thus exceeding the sustained cooling capabilities of the system for short periods. As a result, this scheme is able to improve application responsiveness by up to 6x and save about 30% on power for a conventional Core i7 Desktop chip, as a consequence of finishing computations in a shorter amount of time [20]. This technique was also extended to data centers, in which more interactive workloads such as search or news feeds, that exhibit occasional bursty behavior, can benefit from short performance boosts [25].

Techniques like computational sprinting, that push the limits of what the hardware is designed to do, are conservative with respect to the safety of computations. This is mainly because exceeding hardware specifications leads to system instability. For instance, overclocking or activating a large number of cores for long periods leads to overheating, and exceeding the Thermal Design Power (TDP) of the circuit may cause faults [17], reduce the lifespan, or even physically damage the chip [13]. Hence, after each sprint the CPU ought to switch to a cool-off mode.

Notwithstanding, even outside stable configurations, overclocking has been explored with interesting results. For instance, DSP-accelerators implemented with FPGAs can save up to 39% on hardware resources by overclocking for the same output quality [7]. It was also observed that as the frequency increases the errors in computations gradually appear in the output, up to the point where the program stops producing meaningful results.

In this paper, we envision bringing together these two vectors, by leveraging the redundancy that is already present in a large fraction of data center systems to safely push the limits of computational sprinting in data center environments. In other words, our goal is to aggressively explore overclocking settings, while taking advantage of the existing redundancy introduced by fault tolerance protocols to systematically mask faults that surface, in order to extend the benefits of sprinting (energy efficiency/performance). Furthermore, we intend to explore the synergies and subtle interactions between the two vectors. For instance, the sprints can be coordinated in a way that a subset of the replicas uses a more aggressive but unsafe sprinting to decrease the overall latency, but there is also a sufficient number of non-sprinted replicas that check the results and ensure both availability and correctness in the case of faults.

The remainder of the paper provides an overview of the key techniques that we can leverage as building blocks.

2 COMPUTATIONAL SPRINTING

Advances in CMOS technology have enabled the design of modern multi-core processors, packing an increasing density of transistors at each new generation. However, more transistors implies an increase in power density, at a rate which exceeds the ability to dissipate the heat generated [20]. As a result, continuously operating all the processing units at full power can permanently damage the chip due to overheating. Consequently, some of the cores must remain off most of the time, a limitation known as dark silicon [8].

While it is not possible to activate all the processor cores at once in a sustainable way, there exist proposals for optimizing performance within safe temperature and power limits. These solutions (outlined next), leverage the fact that after activating a sprint, the temperature of the components does not rise instantaneously. Instead, it can take a few seconds for the heat to propagate through the chip package, allowing certain workloads to finish before it overheats.

Parallel sprinting. This approach consists of activating various dark silicon cores for up to a time limit (e.g., 1 sec.) before deactivating cores to cool off. Parallel sprinting can be optimized according to two policies: for maximum responsiveness, it activates all cores at maximum frequency and voltage; for optimal energy efficiency, it activates all cores at minimum frequency and voltage [21]. Parallel sprinting is particularly interesting for mobile devices since a large part of mobile processors are comprised of accelerators that are inactive most of the time. In addition, mobile applications normally have interactive workloads that are characterized by short bursts and long idle times waiting for user input [20].

Note that there is an interesting research question, which we intend to explore, of whether data center workloads also have such characteristics. However, even if they do not, we can still attempt to split replicas in an alternating fashion between a group of replicas that are sprinting and another group that are cooling off.

Frequency Sprinting. A concrete example of frequency sprinting is the dynamic overclocking technology presented in commercial products such as Intel processors with Turbo Boost 2.0 [12]. In a nutshell, Turbo Boost increases the frequencies and voltages of processor cores above a normal safe operation threshold for short periods. The sprint frequency target is defined automatically according to the available resources, allowing the processor to improve performance of both single and multi-threaded workloads. The algorithm that controls when sprinting is activated, takes into account the current frequencies of processor cores, the temperature of the package, and its power consumption [13].

Data center sprinting. Data centers can also experience bursty workloads, e.g., due to shared resources, maintenance activities, garbage collection events, or spikes in service popularity [4, 11]. Therefore, they might be a good match for sprinting. Moreover, the dark silicon phenomenon is likely to be prevalent in data centers, as supported by predictions that stated that by 2024 more than 50% of the chip must be powered off [9]. A current approach called Dynamo [24] employs power capping to enable frequency sprinting while ensuring that the energy drawn remains within the power budget limits. Alternatively, [25] employs coordinated parallel sprinting, using the existing data center backup power supply (e.g. batteries), to provide the extra power for both sprint and cooling.

Overclocking. Pushing hardware to work beyond the prescribed frequency has been studied by other research communities [23]. One approach is increasing the frequency, without sufficiently increasing the voltage, which can lead to faults, because it may violate the propagation delays of circuit critical paths. An interesting characteristic of these fault patterns is that errors gradually appear in the output as frequency increases. This gradual slope in the fault behavior happens because circuit designers are conservative in the estimates of path delays and keep a guard margin between the estimated clock frequency and the reported maximum clock frequency [6], opening the opportunity to explore these limits.

3 FAULT TOLERANCE

Data center systems often rely on distributed replication protocols for operating correctly in the presence of faults. These protocols are designed under certain assumptions about the environment, such as fault behavior (e.g, crash, fail-stop, or Byzantine) and timing (e.g., the synchronous vs. the asynchronous model). Making wrong assumptions about the environment can either put the safety properties of the system at stake or impose an unnecessary cost in terms of replication and consequently energy and infrastructure.

A pragmatic choice for replicating services are asynchronous crash fault tolerant (CFT) protocols [15, 18], because they cover the most common fault and timing behaviors. However, since aggressive overclocking can lead to data corruption or errors in a computation, CFT protocols may be too optimistic for sprinting. Byzantine fault tolerant (BFT) protocols [1, 2, 14] are able to capture data corruption, but they are pessimistic regarding the behavior of faulty replicas. In particular, a BFT adversary may control a fraction of the replicas, allowing collusion, creating the worst case attack scenario. Such pessimism leads to a higher replication costs.

Visigoth fault tolerance (VFT) [19] allows for calibrating the fault tolerance of the system according to the deployment. While it can be configured to capture only crashes with the same replication requirements of CFT, VFT can also capture data corruption with a small additional cost. When compared to BFT, the replication requirements of VFT grows slower as the number of tolerated faults increases. This is because VFT assumes bounded collusion, i.e. the number of replicas that deviate from their expected behavior in the same way simultaneously. One of the reasons why this assumption may be realistic for overclocked systems is that the variations in the chip manufacturing process causes processors to have different stability configurations [7]. Additionally, we can enforce such diversity by carefully controlling overclocking at different replicas.

Other models have been proposed along the same lines as VFT, namely XFT [16], which has the same replication requirements as CFT while capturing Byzantine behavior, although not simultaneously with asynchrony. As future work, we can also explore whether this variant in the set of assumption is met in practice.

4 CONCLUSION

We presented our vision on the potential of combining computational sprinting and fault tolerance to enable higher savings on energy and improved performance for replicated systems. We intend to explore challenges of this approach, by designing and implementing a system that explores the opportunities identified here.

ACKNOWLEDGMENTS

This research is funded by the European Research Council (ERC-2012-StG-307732) and by the FCT (UID/CEC/50021/2013).

REFERENCES

[1] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine Fault Tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI '99)*. 173–186.

[2] Allen Clement, Manos Kapritsos, Sangmin Lee, Yang Wang, Lorenzo Alvisi, Mike Dahlin, and Taylor Riche. 2009. Upright Cluster Services. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles (SOSP '09)*. 277–290.

[3] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. 2012. Spanner: Google's Globally-distributed Database. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI'12)*. 251–264.

[4] Jeffrey Dean and Luiz André Barroso. 2013. The tail at scale. *Commun. ACM* 56, 2 (feb 2013), 74.

[5] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: Amazon's Highly Available Key-value Store. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles (SOSP '07)*. 205–220.

[6] Rui Policarpo Duarte and Christos-Savvas Bouganis. 2012. High-level linear projection circuit design optimization framework for FPGAs under over-clocking. In *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on.* IEEE, 723–726.

[7] Rui Policarpo Duarte and Christos-Savvas Bouganis. 2015. ARC 2014 Over-Clocking KLT Designs on FPGAs Under Process, Voltage, and Temperature Variation. *ACM Trans. Reconfigurable Technol. Syst.* 9, 1, Article 7 (Nov. 2015), 17 pages.

[8] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. 2011. Dark Silicon and the End of Multicore Scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA '11)*. 365–376.

[9] Hadi Esmaeilzadeh, Emily Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. 2012. Dark silicon and the end of multicore scaling. *IEEE Micro* 32, 3 (2012), 122–134.

[10] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. 2007. Power Provisioning for a Warehouse-sized Computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA '07)*. 13–23.

[11] Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, and Alfons Kemper. 2007. Workload Analysis and Demand Prediction of Enterprise Data Center Applications. In *Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization (IISWC '07)*. 171–180.

[12] Intel. 2011. Intel Turbo Boost 2.0. (2011). http://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html

[13] Intel. 2017. 6th Generation Intel Processor Families for S-Platforms Datasheet. (2017). http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/desktop-6th-gen-core-family-datasheet-vol-1.pdf

[14] Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. 2012. CheapBFT: Resource-efficient Byzantine Fault Tolerance. In *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys '12)*. 295–308.

[15] Leslie Lamport. 1998. The Part-Time Parliament. (May 1998). https://www.microsoft.com/en-us/research/publication/part-time-parliament/

[16] Shengyun Liu, Christian Cachin, Vivien Quema, and Marko Vukolic. 2016. XFT: Practical fault tolerance beyond crashes. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*. 1–32.

[17] Edmund Nightingale, John Douceur, and Vince Orgovan. 2011. Cycles, Cells and Platters: An Empirical Analysis of Hardware Failures on a Million Consumer PCs. *6th European Conference on Computer Systems (EuroSys '11)* (2011), 343–356.

[18] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference (ATC'14)*. 305–320.

[19] Daniel Porto, João Leitão, Cheng Li, Allen Clement, Aniket Kate, Flavio Junqueira, and Rodrigo Rodrigues. 2015. Visigoth Fault Tolerance. *10th European Conference on Computer Systems (EuroSys '15)* (2015), 8:1—-8:14.

[20] Arun Raghavan, Laurel Emurian, Lei Shao, Marios Papaefthymiou, Kevin P. Pipe, Thomas F. Wenisch, and Milo M.K. Martin. 2013. Computational Sprinting on a Hardware/Software Testbed. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '13)*. 155–166.

[21] Arun Raghavan, Yixin Luo, Anuj Chandawalla, Marios Papaefthymiou, Kevin P. Pipe, Thomas F. Wenisch, and Milo M K Martin. 2012. Computational sprinting. In *18th Symposium on High Performance Computer Architecture (HPCA '12)*. 249–260.

[22] Malte Schwarzkopf. 2015. *Operating system support for warehouse-scale computing*. Ph.D. Dissertation. University of Cambridge, St John's College.

[23] Kan Shi, D Boland, E Stott, S Bayliss, and G A Constantinides. 2014. Datapath synthesis for overclocking: Online arithmetic for latency-accuracy trade-offs. *51st ACM/EDAC/IEEE Design Automation Conference (DAC '14)* (2014), 1–6.

[24] Qiang Wu, Qingyuan Deng, Lakshmi Ganesh, Chang-Hong Hsu, Yun Jin, Sanjeev Kumar, Bin Li, Justin Meza, and Yee Jiun Song. 2016. Dynamo: Facebook's Data Center-wide Power Management System. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16)*. 469–480.

[25] Wenli Zheng and Xiaorui Wang. 2015. Data Center Sprinting: Enabling Computational Sprinting at the Data Center Level. In *35th IEEE International Conference on Distributed Computing Systems (ICDCS '15)*, Vol. 2015-July. 175–184.